PEDAGOGICKÁ FAKULTA

Univerzita Karlova

Možnosti rozvoje algoritmického myšlení s využitím mobilních dotykových zařízení
Possibilities of developing algorithmic thinking with mobile devices

Martin Kozub

DIPLOMOVÁ PRÁCE

Katedra informačních technologií a technické výchovy

Vedoucí diplomové práce: PhDr. Petra Vaňková, Ph.D.
Studijní program: N7504 Učitelství pro střední školy
Studijní obor: N IT (7504T276)

2019

UNIVERZITA KARLOVA
PEDAGOGICKÁ FAKULTA
Katedra informačních technologií a technické výchovy

ZADÁNÍ DIPLOMOVÉHO ÚKOLU
akademický rok 2017/2018

Jméno a příjmení studenta: **Bc. Martin Kozub**

Studijní program: **N7504 Učitelství pro střední školy**

Studijní obor: Učitelství VVP pro ZŠ a SŠ – informační a komunikační technologie

Název tématu práce v českém jazyce: **Možnosti rozvoje algoritmického myšlení s využitím mobilních dotykových zařízení**

Název tématu práce v anglickém jazyce: **Possibilities of developing algorithmic thinking with mobile devices**

Jazyk práce: **český jazyk**

Stručná charakteristika tématu:
Cílem práce je analyzovat možnosti rozvoje algoritmického myšlení u žáků střední školy se zaměřením na práci s mobilními dotykovými zařízení, zmapovat dostupná řešení a realizovat výzkumné šetření ověřující vybrané řešení rozvoje algoritmického myšlení v edukační realitě.

Zásady pro vypracování:
- Na základě prostudovaných informačních zdrojů zmapujte problematiku rozvoje algoritmického myšlení u žáků střední školy se všeobecným zaměřením.
- Zmapujte možnosti rozvoje algoritmického myšlení pro přímou podporu práce s mobilními dotykovými zařízeními (např. prostředky, prostředí, aplikace).
- Z analyzovaných možností vyberte konkrétní řešení a navrhněte způsoby začlenění do edukačního procesu vzhledem k oblasti Informační a komunikační technologie. Své řešení ověřte v praxi.
- Shrňte výsledky své práce a doporučení pro další praxi.

Předpokládaná struktura práce:
Práci je možné zpracovat v následující struktuře: Úvod - Teoretická a terminologická východiska - Stav poznatků o řešené problematice, její analýza a zhodnocení - Výběr a rozpracování konkrétního didaktického prostředku pro rozvoj algoritmického myšlení - Příprava a realizace výzkumu pro ověření - Zpracování a analýza získaných údajů - Výsledky a jejich hodnocení - Závěry - Seznam použitých informačních zdrojů - Přílohy

Seznam doporučené literatury:
Při řešení budou využívány primární a sekundárních informačních zdroje, včetně elektronických, dle tematické orientace práce.

Vedoucí diplomové práce: **PhDr. Petra Vaňková, Ph.D.**

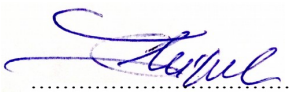
Oponent diplomové práce: **PhDr. Jiří Štípek, Ph.D.**

Předpokládaný rozsah diplomové práce[1]: **60 normostran**

Datum zadání práce: **15. 5. 2018**

Předběžný termín odevzdání práce:[2] **duben 2019**

V Praze dne: 15. 5. 2018



……
PhDr. Jiří Štípek, Ph.D.
vedoucí katedry

Student(ka) stvrzuje podpisem převzetí zadání diplomové práce.

V Praze dne: 15. 5. 2018

……
podpis studenta/studentky

[1] Minimální rozsah diplomové práce je standardně 60 normostran (108 000 znaků vč. mezer) vlastního textu.

[2] Diplomová práce je odevzdávána elektronicky prostřednictvím informačního systému dle harmonogramu akademického roku, zároveň se práce odevzdává v jedné tištěné podobě.

Prohlašuji, že jsem tuto diplomovou práci na téma Možnosti rozvoje algoritmického myšlení s využitím mobilních dotykových zařízení vypracoval pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Dále prohlašuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

V Praze dne 15. 4. 2019

podpis

Poděkování

Rád bych poděkoval vedoucí mé práce PhDr. Petře Vaňkové, Ph.D. za rady a množství času, které mi během konzultací poskytla, stejně jako rodině a přátelům za jejich trpělivost a podporu.

Název práce: Možnosti rozvoje algoritmického myšlení s využitím mobilních dotykových zařízení
Autor: Bc. Martin Kozub
Katedra: Katedra informačních technologií a technické výchovy
Vedoucí diplomové práce: PhDr. Petra Vaňková, Ph.D.

Abstrakt

Práce se zabývá rozvojem algoritmického myšlení u žáků střední školy. V teoretické části se nejdříve orientuje na charakteristiky algoritmického myšlení a analyzuje jeho současný stav a vhodné způsoby postupu výuky. Dále se věnuje programovacím jazykům, metodám programování a soustředí se na možné prostředky (mezi něž patří robotické stavebnice, ale i osobní počítače či mobilní dotyková zařízení) právě pro rozvoj algoritmického myšlení. Od obecných charakteristik směřuje ke konkrétnímu vhodnému didaktickému prostředku - mobilnímu dotykovému zařízení - a možnostem programování s ním, resp. pro něj, což se stává i hlavním východiskem pro empirickou část práce. Ve zvoleném akčním výzkumu jsou nejdříve navrženy vhodné aktivity a postup pro rozvoj algoritmického myšlení. Tato koncepce byla následně ověřena v praxi na střední škole. Výsledky výzkumného šetření jsou shrnuty i v podobě doporučení pro další praxi.

Klíčová slova: algoritmické myšlení, výuka informatiky, dekompozice úloh, blokové programování

Title: Possibilities of developing algorithmic thinking with mobile devices
Author: Bc. Martin Kozub
Department: Katedra informačních technologií a technické výchovy
Supervisor: PhDr. Petra Vaňková, Ph.D.

Abstract

This thesis explores algorithmic thinking development on a high school student level. In the introduction algorithmic thinking as well as current practices of teaching algorithmic thinking are examined. A closer look on programming languages, programming methods and tools which can be utilised to develop algorithmic thinking (such as robotic kits, personal computers and mobile devices with touch screens) follows. After analyzing general properties of these tools one mobile device is selected and its programming possibilities are then explored in the empirical part of this paper. As part of the selected proactive action research a number of activities as well as overall process approach is first designed following by their practical verification in a high school. Final results are summarised as suggestions for improvements for further teaching.

Keywords: algorithmic thinking, ICT education, problem decomposition, block progamming

Obsah

1 Úvod

S globálně rostoucím významem informačních technologií ve všech aspektech života vzniká i stále větší potřeba zabývat se tímto tématem a problémům s ním spojených také na školách. Nejde ale pouze o to navyšovat informačním technologiím časovou dotaci, nýbrž především o potřebu upravit výuku tak, aby byla, co se užitečnosti žákům týče, vhodnější. (Noonan, *The Growing Importance of Computer Science*)

Výuka informačních technologií na českých školách se dnes potýká s mnoha problémy, ať již je to nepřipravenost a nízká kvalifikace učitelů, tak také s tím související neadekvátní průběh samotných hodin. Mnoho let se tak na většině základních i středních škol v hodinách ICT vyučuje kromě nezbytné teorie především ovládání uživatelského rozhraní, a co hůře navíc konkrétních zástupců softwaru. Dnešní trend, který je možné pozorovat nejen v rozvinutějších zemích světa a s určitým zpožděním také u nás, ale přechází právě od tohoto přístupu k obecnějšímu pojetí, přičemž klade důraz na pochopení samotných principů a schopnost s nimi pracovat a reprodukovat je pro další použití. (Paulenková, "Výuka informatiky na školách se mění, zaměří se na programování")

S tím souvisí primárně potřeba rozvíjet schopnosti myšlení žáků takovým způsobem, aby přijali tyto principy za vlastní. Protože se jedná o aktuální téma, jímž je potřeba se ve školství aktivně zabývat, soustředí se tato práce právě na algoritmické myšlení, které umožňuje řešení komplexních úloh jejich rozložením na jednotlivé části. V tomto směru je zaměřena nejen na samotný pojem a jeho význam včetně souvislostí, ale zvláště na jeden z možných způsobů samotného rozvoje algoritmického myšlení s důrazem na žáky středních škol.

Diplomová práce vychází z návrhů pro nadcházející rámcově vzdělávací programy (RVP), které algoritmizaci jako takovou již explicitně zahrnují. (*Školská informatika končí, děti se budou učit programovat. Učitelé však nejsou na převratné změny připraveni*) Snaží se navrhnout postup, jakým lze výuku této problematiky realizovat. K tomuto účelu byla v širokém spektru možností zvolena dotyková zařízení, která poskytují nejrůznější prostředky, jako jsou senzory či připojení k internetu, s jejichž pomocí je možné dosáhnout větší variability, tj. více možností realizace a tím pádem i většího prostoru pro tvořivost žáků. Vynikají ale i interaktivitou a rychlostí nasazení, zvláště v porovnání s robotickými stavebnicemi.

Cílem práce je zjistit, zda mohou mobilní dotyková zařízení v tomto směru pomoci a zda je jejich využití ve výuce vhodné a vede ke zlepšení digitálních kompetencí žáků. Předmětem výzkumu budou většinoví žáci středních škol.

2 Vymezení problému

Hlavní oblastí, kterou se práce zabývá, je možnost využití mobilních dotykových zařízení v rámci výuky algoritmizace. Především pak zjistit, zda jejich využití vede v edukačním procesu k dosažení osvojení algoritmického myšlení žáky, a tudíž zda je jejich aplikace ve výuce přínosná.

S ohledem na definovaný hlavní problém je možné vyčlenit několik souvisejících dílčích problémů, které je možné formulovat do otázek následovně:

P1 Jaká jsou teoretická východiska algoritmického myšlení?

P2 V jakém stavu je výuka algoritmického myšlení především na českých školách?

P3 Jakým způsobem je možné realizovat výuku algoritmického myšlení?

P3.1 Jaký je vhodný průběh výuky algoritmického myšlení?

P3.2 Jaké prostředky je možné k výuce obecně použít?

P3.3 Jaký styl programování[1] je s ohledem na komplexnost vhodné využít?

P3.4 Jaké možnosti se nabízejí konkrétně při využití mobilních dotykových zařízení?

P4 Je zvolený postup implementace do výuky pro rozvoj algoritmického myšlení vhodný?

3 Cíle práce

Primárním cílem diplomové práce je ověřit, zda mohou mobilní dotyková zařízení a jejich využití při výuce algoritmizace pomoci v edukačním procesu.

Hlavní cíl je možné rozdělit do těchto dílčích cílů:

C1 Popsat teoretická východiska algoritmického myšlení a zjistit jeho přínosy.

C2 Analyzovat současný stav na středních školách, které výuku algoritmického myšlení realizují.

C3 Specifikovat didaktické prostředky vhodné pro rozvoj algoritmického myšlení.

C3.1 Popsat průběh výuky algoritmického myšlení.

C3.2 Analyzovat zařízení, která je možné při výuce algoritmického myšlení použít.

C3.3 Charakterizovat programovací jazyky a prozkoumat možnosti tvorby programů pro podporu výuky na střední škole.

C3.4 Analyzovat možné nástroje pro využití mobilních dotykových zařízení ve výuce na středních školách.

C4 Realizovat výzkumné šetření pro podporu rozvoje algoritmického myšlení na střední škole.

[1] Stylem programování je myšlen výběr z celé škály dostupných programovacích jazyků, včetně volby programovacího prostředí a především pak rozhodnutí mezi klasickým psaním kódu a skládáním grafických bloků.

4 Výzkumné metody

K dosažení cílů uvedených v předchozí kapitole je potřeba provést nejprve teoretickou analýzu primárních a sekundárních zdrojů, převážně pak elektronických. V oblasti algoritmického myšlení je pak zvláště potřeba věnovat se materiálům didaktickým, které mohou odpovídat na otázky vyplývající z cílů, především pak z cíle C3.

V návaznosti na teoretický základ a jeho interpretaci bude vytvořen projekt sestávající z dílčích návrhů na tvorbu aplikací pro mobilní dotyková zařízení. V praxi pak bude ověřena přínosnost za pomoci empirického výzkumu.

Výzkum bude realizován pomocí proaktivního akčního výzkumu podle R. A. Schmucka, který staví na vývoji nových postupů a následně vyhodnocuje efektivitu: „V rámci tohoto přístupu se pokoušíme o nové přístupy přinášející lepší výsledky, pravidelně sbíráme informace a shromažďujeme a vyhodnocujeme reakce na změnu, pokoušíme se o nové alternativní přístupy.“ (Pavelková, “Akční výzkum v pedagogickém prostředí”)

Výzkum bude sestávat z následujících kroků:

1. **Zavedení nových postupů** ve snaze dosáhnout lepších výsledků
   Aplikace výuky algoritmického myšlení s využitím mobilních dotykových zařízení.
2. **Shrnutí předpokladů výsledků**, zahrnující jak pozitivní, tak negativní výstupy.
3. **Průběžné získávání zpětné vazby od žáků**
   Zpětná vazba bude získávána v průběhu jednotlivých lekcí na základě reakcí žáků na probírané učivo a jeho výuku prostřednictvím vybraného nástroje. Také pozorování učitele v průběhu může odhalit některé obtíže a bude nedílnou součástí celkového hodnocení. Závěrečné hodnocení pak proběhne formou požadavku na zpracování komplexní úlohy s aplikací základních algoritmických konceptů.
4. **Revize a návrhy na zlepšení**
   V souvislosti se zpětnou vazbou bude dále provedena reflexe. Ze závěrečného hodnocení pak vyplynou případné obtíže a možné návrhy na zlepšení. (*Akční výzkum ve škole*)

Část I

Teoretická východiska

Na základě vymezených cílů je nezbytné prozkoumat pojmy algoritmus a algoritmické myšlení (C1) a analyzovat stav výuky algoritmického myšlení na středních školách (C2). Rovněž je třeba se zaměřit na možná východiska pro realizaci této výuky a to především z pohledu průběhu vyučovacího procesu (C3.1), dostupných programovacích jazyků (C3.3), nástrojů vhodných pro vývoj (C3.4) a v neposlední řadě také hardwarového vybavení, které je možné využít jako didaktické pomůcky (C3.2).

5 Algoritmus

Pojmem algoritmus se myslí přesný popis procesu, vedoucího k řešení úlohy. (*Algoritmus*) Nejtypičtějším příkladem jeho užití je programování, jež vyžaduje nalezení algoritmu k následnému zápisu do zdrojového kódu programu. (*Imperativní programování*) Algoritmický postup ovšem není výhradou programování a vyskytuje se zcela běžně také při nejrůznějších dalších činnostech. Za zmínku stojí jednoznačně matematika, kde obecná znalost řešení problému vede vždy k získání správného výsledku.

K tomu, abychom postup mohli za algoritmus označit, ale musejí být současně splněny všechny následující podmínky:

1. **Konečnost** – Algoritmus musí skončit v omezeném množství kroků, nemůže být nekonečný.
2. **Obecnost** – Algoritmus musí být univerzální, nemůže řešit jen jeden konkrétní problém. Musí tedy například zvládnout řešit obecně sčítání dvou čísel, nikoliv pouze konkrétních. K tomu bychom nepotřebovali algoritmus, ale postačily by tabulky.
3. **Determinovanost** – V každém kroku musí být jednoznačné, co bude následovat. Platí tedy, že pro stejné vstupy získáme vždy stejné výstupy.
4. **Resultativnost** – Algoritmus má vždy alespoň jeden výstup, čímž poskytuje odpověď na konkrétní problém.
5. **Elementárnost** – Obecně platí, že jednotlivé kroky algoritmu mají být co nejjednodušší. Tzn., že bychom v algoritmu neměli shlukovat jednotlivé bloky procesů do jednoho kroku, ale i ty dělit na části.

Výše uvedený výčet požadavků na algoritmus je do jisté míry zavádějící, neboť není univerzálně platný. Některé zdroje uvádějí také další kritéria, naproti tomu jiné zase některé postrádají. Nicméně mnoho zdrojů kolem tohoto seznamu těsně osciluje a lze ho tak považovat za dostatečný. (*Algoritmus*)

V návaznosti na uvedené požadavky na algoritmus lze tvrdit, že například schopnost sečíst čísla 5 a 5 není algoritmem, neboť zde není splněna podmínka obecnosti. Naopak obecný postup, jak sečíst libovolná dvě čísla, již algoritmem je, protože není omezen na jeden specifický problém, ale dokáže pracovat s libovolnými vstupními hodnotami.

Elementárnost

Zatímco požadavky 1 až 4 jsou jednoznačné, elementárnost jako taková je do jisté míry relativním pojem. To, co může být pro někoho jednoznačné, nemusí být někomu jinému vůbec zřejmé. Rozdíly jsou patrné například mezi žáky různých věkových kategorií.

Operace, které jsou pro žáka ještě elementární, je možné zjistit pouze experimentálně. „Provedou-li podle návodu všichni žáci určitou operaci správně a jednoznačně, pak je možno říci, že tato operace je pro ně elementární. Jestliže ji nevykonají správně a jednoznačně, pak není elementární.“, tvrdí L. N. Landa. (Landa, *Algoritmy a učení*, s. 24)

Vždy tedy záleží na tom, v jaké rovině se s algoritmy pracuje. Pokud jsou vytvářeny algoritmické úlohy pro žáky, je potřeba reflektovat jejich schopnosti. Analogicky, jestliže je cílem vytvořit počítačový program, je nutné pracovat s takovými operacemi, kterým počítač rozumí. Elementárnost se může v tomto směru odvíjet například od použitých knihoven, které mohou vyvíjenému programu umožnit ve svém kódu využívat prvky vyšších úrovní, než v případě absence těchto knihoven. To znamená, že s patřičnou knihovnou lze například vytvořit z operace „najdi největší číslo v poli“ elementární operaci, pro kterou by bylo v některých případech nezbytné psát vlastní implementaci. Ačkoliv elementární operace sama uvnitř sestává z většího počtu dílčích kroků a pro konečné zpracování počítačem se tak nic nemění.

6 Algoritmické myšlení

Algoritmické myšlení, jakožto podmnožina informatického myšlení, je účinný způsob myšlení, který obecně usnadňuje hledání řešení problémů. Jedná se o dovednost užitečnou, ne-li nezbytnou, při programování či zpracovávání dat.

Toto myšlení tvoří jeden ze základních konceptů počítačové vědy, v českém jazyce typicky označované za matematickou informatiku. (*Matematická informatika*) Jedná se o nepostradatelný nástroj sloužící k řešení problémů, který našel své aplikace nejen ve vědě, ale i za jejími hranicemi. Perspektivní výuka informatiky by se tak měla opírat právě o algoritmické myšlení. (Hromkovič et al., *Examples of Algorithmic Thinking in Programming Education*)

Algoritmizace staví na teorii deterministického řízení. Odsud vyplývá, že „všechna řešení mohou být rozčleněna do algoritmických, poloalgoritmických, heuristických a poloheuristických skupin“. Pro řešení algoritmických problémů je nezbytné využití struktury uspořádaných kroků a předem definovaný postup. To znamená, že pro ně je možné definovat jednoznačný sled instrukcí. Naopak pro řešení heuristických problémů je nezbytné, vzhledem k neobvyklosti situace, kreativní myšlení. Kreativní postup ale není možné algoritmem popsat. (Rambousek, *Edukační technologie*)

Pokud se žáci drží algoritmického postupu a následují sled operací, je zaručeno, že se doberou ke správnému řešení. Podle L. N. Landy lze ve výuce nalézt celou řadu úloh algoritmické povahy, pro které příznačně využívá termínu „výukový algoritmus“. A protože při dodržování algoritmických postupů nedochází k omylům, obejde se i učení bez slepých uliček. (Skalková, *Obecná didaktika*, s. 158)

L. N. Landa zvláště zdůrazňuje potřebu takovýto postup dostatečně konkretizovat tak, aby byl každý krok zcela jednoznačný. Uvádí pro to v kontrastu následující příklady:

„Představme si, že metodik ve snaze naznačit učiteli, jak má učit žáky řešit matematické úlohy, takto popisuje průběh řešení:

Aby bylo možno vyřešit úlohu, je třeba

1. pochopit podmínky;
2. rozdělit úlohu na hlavní prvky;
3. zjistit jejich vztah;
4. uvést prvky do systému;
5. vícekrát prozkoumat prvky, ale z různých hledisek;
6. pokusit se najít něco známého v tom, co je předmětem zkoumání a něco užitečného v tom, co se ukázalo být známým atp.“

Výše citovaná ukázka je poměrně extrémním příkladem, jak by popis neměl vypadat. Uvedený model řešení matematické úlohy totiž nerozděluje jednotlivé kroky na dostatečně elementární operace. L. N. Landa dále popisuje konkrétní problémové části. „Co to znamená pochopit podmínky? Co je pro to potřeba udělat? Říci ‚pochopit podmínky‘ a neříci jak to udělat – to je totéž, jako neříci nic. V návodu se dále říká, že je třeba úlohu rozdělit na hlavní prvky a zjistit jejich vzájemný vztah. Ale které prvky úlohy mají být považovány za hlavní a co znamená ‚zjistit jejich vztah‘? Ze kterých konkrétních operací se skládá ‚zjištění vztahu‘, co je třeba konkrétně udělat, aby byl zjištěn vztah? O tom se v návodu nic neříká. Tytéž nedostatky mají i všechny ostatní pokyny, které návod obsahuje.“

Naproti tomu stojí příklad autorem označovaný za správný:

„Když někomu řekneme, aby písemně sečetl dvě různá čísla, je třeba:

1. napsat čísla pod sebe tak, aby poslední číslice byly pod sebou;
2. sečíst poslední číslice;
3. jestliže je součet menší než deset, napsat toto číslo a přejít ke sčítání dalšího sloupce;
4. jestliže je součet větší než 10, napsat poslední číslo, zapamatovat si číslo první a přejít ke sčítání dalšího sloupce… atd.“

Pokud je popis dostatečně konkrétní a jednoznačný, stačí žákovi rozumět jazyku, ve kterém je popis napsán, a bude schopen úlohu s jeho pomocí vyřešit. (Landa, *Algoritmy a učení*, s. 24-25)

Článek, publikovaný v SIGCSE 2018 (Technical Symposium on Computer Science Education) (*SIGCSE Technical Symposium on Computer Science Education*), definuje algoritmické myšlení jako schopnost navrhovat, implementovat a hodnotit implementaci algoritmů za účelem řešení různých problémů. Zahrnuje identifikaci a pochopení problému, úpravu algoritmu či více algoritmů pro řešení určitého problému, implementaci algoritmů takovým způsobem, že řeší daný problém a vyhodnocování implementace na základě kriterií. (Thomas, *ACM Digital Library*, s. 149-154)

O algoritmizaci, jako jedné z klíčových kompetencí, hovoří také rámec digitálních kompetencí občanů DigComp 2.0 (The Digital Competence Framework). V kategorii „Tvorba digitálního obsahu“ uvádí jako jeden z podbodů programování, kde definuje požadavek na schopnost plánovat a vytvářet takové sekvence srozumitelných instrukcí pro výpočetní systém, které budou řešit předložený problém nebo provádět určitou úlohu. (Vuorikari et al., *DigComp 2.0: The Digital Competence Framework for Citizens*, s. 9)

Paul Curzon definuje algoritmické myšlení jako cestu k získání řešení problému skrze jednoznačně definované kroky. Nic se nestane jen tak. Raději, než odpověď na konkrétní specifický problém, vyvíjejí žáci algoritmus. Existují instrukce či pravidla, které pokud jsou přesně dodrženy, vedou k řešení původního, ale i podobných problémů. Pokud člověk či stroj zná metodu násobení, může vynásobit libovolná dvě čísla. Nemusí se pokaždé od počátku zaobírat tím, jak funguje násobení. Silnou stránkou algoritmického myšlení je, že umožňuje automatizaci hledání řešení. (Curzon, *Algorithmic Thinking*)

Gerald Futschek popisuje algoritmické myšlení jako termín, který je velmi často uváděn jako jedna z nejdůležitějších kompetencí, které lze výukou v předmětu informatika dosáhnout. Jedná se o soubor dovedností, které jsou společně propojeny a vedou ke konstrukci a analýze algoritmů:

- schopnost analýzy problému
- schopnost přesně definovat (popsat) problém
- schopnost vytipovat základní metody, které jsou pro řešení souvisejícího problému užitečné
- schopnost sestavit a opravit algoritmus pro řešení určitého problému, s použitím základních metod
- schopnost uvažovat všechny možné speciální i běžné situace, které mohou nastat
- schopnost vylepšit efektivitu algoritmu (Futschek, *Algorithmic Thinking: The Key for Understanding Computer Science*)

Krátkou, ale výstižnou definici, dále uvádí také uvádí průvodce pro učitele CAS Computational Thinking. Algoritmické myšlení je způsob, jak dojít k řešení pomocí jasně definovaných kroků. (Csizmadia et al., *Computational thinking*, s. 7)

6.1 Znaky algoritmického myšlení

Na základě kurzů Houstonské univerzity Rice sestává algoritmické myšlení z hlavních pěti na sebe navazujících kroků:

1. Obdržení problémové úlohy a její pochopení
   (a) Znalost odpovědi na otázku, jaká data obdržím.
   (b) Znalost odpovědi na otázku, jaký výstup je očekáván, co je předmětem zkoumání.
2. Formulace problému z hlediska vstupu a výstupu.
   (a) Cílem tohoto kroku je přepis vstupních dat do takové podoby, které počítač rozumí. To vyžaduje matematické schopnosti a též znalost formálního jazyka k reprezentaci vstupu programu.
   (b) Druhá část spočívá opět ve formální definici výstupu, který budeme od programu očekávat. Je si tedy potřeba říci, co budeme ve vstupních datech hledat.
3. Návrh algoritmu k samotnému řešení problému. Jeho vstupem budou data formálně zpracovaná v předchozím kroku a jeho výstupem formát opět odpovídající definovanému výstupu, známému též z předchozího kroku. Během tvorby algoritmu je dále potřeba soustředit se na jeho správnost a výpočetní požadavky, tj. například zda je možné na něj aplikovat i větší objemy dat.
4. Implementace algoritmu. Tento krok vyžaduje znalosti programovacího jazyka a zkušenosti pro správný přepis návrhu algoritmu do programovacího jazyka. Během přepisu totiž může

dojít k chybám či problémům s efektivností. Na druhou stranu je možné využít znalosti programování ke zrychlení i nad rámec toho, co algoritmus teoreticky garantuje.

5. Spuštění na datech a odpověď na původní otázku. V posledním kroku je potřeba si připomenout, kde vše začalo. Je potřeba použít vytvořený program na datech, která byla na začátku poskytnuta, a vrátit požadovaný výsledek. (Nakhleh, *What is Algorithmic Thinking?*)

Pro přiblížení výše uvedeného výpisu lze použít například tento postup:

1. Zadání problémové úlohy: zjistěte, v kolika procentech případů platí, že pokud se osoba A přátelí s osobou B a osoba B s osobou C, jsou i osoba A a C přátelé. Pro řešení úlohy budou dostupné údaje z některé ze sociálních sítí.
2. K dispozici je tedy tabulka s informacemi. Bude užitečné ji transformovat například do podoby grafu, kterou bude budoucí program moct dobře číst. Výstupem bude četnost případů, kdy platí, že existuje-li ve vstupním grafu vazba mezi A a B i B a C, tak existuje také vazba A a C.
3. Sestavte teoretický model, jak budete postupovat od vstupu k výstupu. Měli byste se zabývat také jeho správností a optimalizací.
4. Navržený algoritmus přeneste do zdrojového kódu programu.
5. Vytvořený program spusťte na vstupních datech a výsledek předejte například zadavateli. (Nakhleh, *What is Algorithmic Thinking?*)

6.1.1 Opozice

Kromě zastánců algoritmického myšlení především v souvislosti s jejím zařazením na školách existuje i poměrně početná skupina lidí, která tvrdí, že by algoritmizace neměla být vůbec součástí vyučování, ale výuka by se měla soustředit primárně na objektově orientované programování (OOP). (Neumajer, *Ivan Ryant: Nahraďte algoritmizaci systémovým přístupem!*) Svůj názor staví především na tom, že v komerční sféře již není tak důležité znát základy programování, tj. například být schopen vytvořit algoritmus pro seřazení prvků v poli, ale stačí je umět použít s některou z dostupných knihoven. Mezi přednosti OOP pak uvádějí zejména větší produktivitu, nižší chybovost, větší přehlednost kódu a s tím související snazší údržbu a spolupráci napříč týmem.

Ačkoliv se zastánci tohoto názoru jistě nelze nesouhlasit v tom smyslu, že by snad běžný programátor musel neustále vymýšlet nové algoritmy pro již nespočetněkrát řešené problémy, pro které existují v ideálních případech nejrůznější implementace, z nichž si lze vybírat. Není jim možné upírat skutečnost, že objektově orientované programování je snad již považováno za lepší alternativu klasických strukturovaných programů, je nutné si uvědomit, že škola zde není pouze pro komerční sféru, ale především pro to, aby žákům předala znalosti a donutila je nad věcmi přemýšlet. Proto je nesmírně důležité neopomíjet ani algoritmizaci, bez které žáci neporozumí základním principům a budou s voláními knihovních funkcí pracovat naprosto bez povědomí o tom, co se v nich může odehrávat. V podstatě se pak z programování stane práce s černými skříňkami, což je mj. smyslem OOP, ovšem zcela v jiném kontextu než kritici výuky algoritmizace naznačují. Černou skříňku v kontextu zpřístupněného rozhraní pro komunikaci s objektem není možné zaměňovat s ignorantstvím, tj. neznalostí principů.

Dnešní výuka programování na školách, včetně vysokých škol, se opírá právě o algoritmické myšlení a teprve po jejím úspěšném absolvování zařazuje blok objektově orientovaného programování.

V případě středních škol ale na tuto druhou část již typicky nezbývá mnoho času, je-li vůbec zařazen, což je mj. předmětem kritiky opozice.

V souvislosti s poznatky o algoritmizaci i objektově orientovaným programováním je jednoznačně potřeba oba aspekty vyvážit a zařadit do výuky tak, aby se staly oba přístupy žákům povědomé na takové úrovni, aby s nimi dokázali pracovat. K tomu je potřebné uvažovat nad tím, jakým způsobem vlastně tyto způsoby do výuky zařadit a jak je studentům prezentovat.

O tom, zda výuku pojmout tak, aby začínala prací s objekty a postupně se dostávala k nižším úrovním, tedy konkrétním algoritmům, nebo naopak, aby šla od celků k jednotlivostem, se patrně ještě dlouho povedou diskuze. Nicméně by v žádném případě neměla být opomenuta ani jedna část. Přesto je patrné, že by se v rámci algoritmizace nemělo soustředit na tvorbu komplexních programů, ale pouze na jednotlivá řešení konkrétních jednodušších problémů. V případě práce na něčem větším by pak škola měla vést žáky primárně k využívání objektově orientovaného přístupu:

„Když už se žáci naučí programovat strukturovaně, odmítají následně změnit způsob svého uvažování. Jejich programy proto často nejsou objektově orientovanými programy, ale pouze strukturovanými programy používajícími třídy. A to je citelný rozdíl.“ (Pecinovský, *Metodika výuky programování na rozcestí*)

Navzdory výše uvedenému je však nezbytné nezapomínat ani na existenci potřeby nízkoúrovňových vývojářů, jejichž práce spočívá ve tvorbě firmwarů a jednoúčelových programů pro nejrůznější čipy a mikropočítače, kde je potřeba zabývat se optimalizací zásadní, a méně čitelný ale rychlejší kód nebo kód vyžadující menší množství hardwarových prostředků má přednost. To pouze potvrzuje, že není možné generalizovat algoritmizaci jako něco v praxi nepoužitelného či překonaného a je potřeba do vzdělávání zahrnout oboje.

6.2 Výuka algoritmického myšlení

Výuku programování a tudíž i algoritmického myšlení lze podle J. Hromkoviče a spol. rozdělit do tří návazných sekcí:

1. Schopnost práce s programovacím jazykem, jakožto formálním jazykem pro zápis algoritmu.
2. Abstrakce a automatizace jako základní strategie pro řešení problémů.
3. Praktická omezení spočitatelnosti jako motivace pro vylepšování existujících algoritmů.

6.2.1 Formální jazyk

Studentům je programování představeno jako prostředek pro sdělování instrukcí počítači. Z počátku je sada instrukcí velmi omezená, navíc má každá instrukce přesně definovanou syntaxi i sémantiku, aby se předešlo možným dvojznačnostem. Protože prvotní slovník není obsáhlý a nepokrývá komplexnější úlohy, jsou žáci brzy nuceni vytvářet si nová slova, za nimiž se skrývá sada podprocesů.

Typickou ukázkou v tomto kontextu je programovací jazyk Logo a želva pohybující se na monitoru počítače. Pod základními kroky si lze představit například otočení či pohyb vpřed. Želvička za sebou může zanechávat stopu a pokud s ní bude cílem nakreslit nějaký obrazec, potom je nezbytné

ji vézt po určité trase konečným počtem kroků sestávajících z příkazů otočení, pohybu vpřed atd. Pro usnadnění práce je ale možné zabalit řadu instrukcí do jednoho balíku tak, jak jdou za sebou, a volat jej s novým jménem. Toto je užitečné zejména v případě, kdy je některé části nezbytné vícekrát opakovat.

Tak se studenti v praxi setkají s koncepty jako jsou modularizace, formální jazyk a vyjádření významu v algoritmické podobě.

6.2.2 Abstrakce a automatizace

Programování ale není jen o skládání instrukcí. Například modularizace začíná odkrývat svůj plný potenciál až v momentě, kdy jsou k ní přidány proměnné. Zatímco mít vlastní obslužnou metodu pro nakreslení čtverce usnadňuje pochopení toho, co program v určitém místě dělá, s přidanou proměnnou ovlivňující velikost se z ní stává metoda univerzální, použitelná i pro případy, na které její autor nemusel myslet. Přidáním dalšího parametru je dosaženo ještě větší abstrakce – například určením počtu vrcholů lze dosáhnout funkce, s jejíž pomocí je možné nakreslit libovolný mnohoúhelník.

V tomto ohledu se pak žáci seznámí také s cykly. Například u zmíněného mnohoúhelníku se bez cyklů v podstatě obejít nedá. I pokud by byl počet vrcholů znám předem, je bez cyklů tvorba příslušné obslužné metody otravná v tom smyslu, že vyžaduje opakování celků kódu.

Automatizace ani abstrakce ale nejsou koncepty specifické pouze pro programování, nýbrž se uplatňují v matematické informatice a algoritmickém myšlení v širším slova smyslu. Jakmile je řešení jednoho konkrétního případu známo, lze použít automatizaci na všechny případy stejného problému.

6.2.3 Praktická omezení spočitatelnosti

Automatické nalezení řešení není vždycky možné. Skutečnost, že existují problémy, které nejsou algoritmicky řešitelné, ukázal ve svém referátu „On Computable Numbers, with an Application to the Entscheidungsproblem“ Alan Turing již v roce 1937 (Turing, “On Computable Numbers, with an Application to the Entscheidungsproblem”, s. 230-265). Ale ani spočitatelné problémy nemají vždy snadné řešení a mohou vyžadovat obrovské a v reálném světe nedostupné množství výpočetních prostředků, nebo je potřeba zaokrouhlovat kvůli numerickým chybám.

Ve vzdělávání je ale potřeba tuto oblast zviditelnit, aby byla pro studenty hmatatelná. Ve zmíněném nástroji s želvičkou pohybující se po obrazovce je vhodné využít například kruhu, který počítač nikdy nemůže nakreslit zcela přesně. Zde se využívá aproximace s použitím mnohoúhelníku. V případě jeho využití na místo kruhu je potřeba zvolit kompromis mezi velice jemným mnohoúhelníkem a hrubým mnohoúhelníkem a to proto, aby se pozorovateli jevil útvar po okraji dostatečně hladký a přitom nedocházelo k zásadním výpočetním problémům, neboť na každém vrcholu se musí želvička pootočit a posunout k dalšímu vrcholu. Čím menší je tedy počet těchto kroků, tím rychlejší je proces výpočtu. Žáci se s tímto aspektem musejí vypořádat, ovšem záhy narazí na omezení samotné obrazovky, tedy jeho rozlišení, které přesnější replice kruhu stejně nenechají vyniknout. (Hromkovič et al., *Examples of Algorithmic Thinking in Programming Education*, s. 14)

6.3 Alternativní pohled na výuku algoritmizace

Zatímco J. Hromkovič považuje znalost formálního jazyka za nezbytnou prerekvizitu veškerého dalšího snažení, byť začíná s velmi omezenou sadou instrukcí, je podle zkušenosti G. Futscheka možné úvod do algoritmů vyučovat čistě na úrovni problému, nezávisle na jakémkoliv programovacím jazyce i jeho znalosti. Souhlasí ale, že vizualizace algoritmů v nějaké formě je nezbytná. Jako alternativu ale pro naprosté začátečníky uvádí například aktivizující hry, v nichž žáci sami plní roli algoritmu (stávají se nástrojem na místo programovacího jazyka). Rovněž ale zdůrazňuje nezbytnost znalosti abstraktního modelu, včetně přesných definic jeho vlastností, pro zlepšování a modifikaci algoritmů. (Futschek, *Algorithmic Thinking: The Key for Understanding Computer Science*, s. 159-168)

6.4 Stav výuky algoritmického myšlení u nás a ve světě

Výuka algoritmického myšlení je úzce spjata s programováním, které se k tomuto účelu na školách využívá. Cílem běžně zařazených hodin programování na školách by tedy nemělo být primárně naučit žáka programovat v konkrétním programovacím jazyce, i když i to je patrně nevyhnutelným a neškodným výstupem těchto hodin, ale hlavní důraz by měl být kladen na rozvoj algoritmického myšlení žáků.

Programování na školách ale není žádným novým trendem, nýbrž něčím, o čem se vedou debaty a co se reálně praktikuje již dlouhá desetiletí. K těmto účelům vznikly i speciální programovací jazyky, které jsou snadno pochopitelné (využívají například reálná slova z běžného mluveného jazyka) a názorné (disponují grafickým výstupem znázorňujícím chod programu). Typickým příkladem je programovací jazyk Logo, za jehož zrodem stojí Wally Feurzeig, Seymour Papert a Cynthia Solomon, a který vznikl již v roce 1967. (*Logo (programming language)*) Zástupců je ale mnohem více – například Karel[2], Kodu[3] nebo Scratch[4]. (Williams, *Introducing computing*, s. 68)

Například uvedený jazyk Scratch spadá do kategorie jazyků, kde uživatel nepíše program jako textový seznam, což je typické u mnoha jiných jazyků, ale skládá jednotlivé bloky kódu v klikatelném grafickém prostředí. Protože skládání kódu funguje podobně jako puzzle v tom smyslu, že je potřeba najít pár, který do sebe zapadá, aby na příslušné místo bylo možné nový blok vložit, je nemožné v takovém prostředí udělat chybu v syntaxi.

Zatímco s programování v jazycích jako je Baltík[5] nebo Karel, které vedou k ovládání scény na obrazovce počítače, se bylo možné na školách setkat i dříve, dnešní trend ukazuje spíše cestu vedoucí skrze skutečné hardwarové roboty, kteří, ač mohou dělat stejné věci, jsou pro žáky atraktivnější a názornější. (Schön, *Programování se má stát běžným předmětem ve škole, s výukou pomáhají roboti. Předmět naráží překvapivě na odpor rodičů*)

K tomu, aby byl takový robot ve vzdělávání vhodný, musí být splňovat řadu podmínek – především musí být práce s ním bezpečná, dále musí být robot jednoduše použitelný, ne zbytečně komplikovaný

[2] *Karel (programovací jazyk)* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/Karel_(programovac%C3%AD_jazyk).

[3] *Běhové prostředí* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2017 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/B%C4%9Bhov%C3%A9_prost%C5%99ed%C3%AD.

[4] *Scratch* [online]. Massachusetts: MIT Media Lab, 2018 [cit. 2018-12-16]. Dostupné z: https://scratch.mit.edu.

[5] *Baltík* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/Balt%C3%ADk.

a měl by vydržet horší zacházení. Důležitým faktorem je určitě i atraktivita, tak aby děti měly vlastní zájem se zařízením pracovat. A v neposlední řadě hraje roli také cena.

Na rozdíl od typických hodin informatiky na některých českých školách, kde probíhá výuka práce s kancelářskými balíky jedné konkrétní firmy (Endrštová, *Nový způsob výuky informatiky nemá děti učit Word a Excel, ale programování a stavění robotů. Učitelé se ho ale bojí*), přinášejí robotické programovatelné hračky větší potenciál na rozvoj tvořivého myšlení. V rámci edukační robotiky řeší student určitý problém, který se musí naučit vhodně rozdělit na menší části, a ty pak s použitím dostupných prostředků vyřešit.

Výuka programování by neměla být výhradní výsadou technických škol, ale měla by se objevovat napříč celým spektrem. Hledání nejlepšího řešení je univerzální problém při mnoha aktivitách a takto získané dovednosti jsou vhodné i pro žáky humanitních a ekonomických oborů, kde je však aktuálně míra znalostí z této oblasti podstatně horší. (Černý, *Výukoví roboti: nástroj pro rozvoj algoritmického myšlení*)

Některé školy, které programování standardně nevyučují, se alespoň hlásí do programů jako je mezinárodní kampaň Hodina kódu od organizace Code.org, jež podporuje i celá řada soukromých firem, nebo i ryze český Bobřík informatiky. (*Stovky českých škol zařadily do své výuky programování*)

Zatímco některé jiné země Evropy, například Estonsko (Paulenková, "Výuka informatiky na školách se mění, zaměří se na programování") či Velká Británie, zahrnuly výuku programování do svých programů, u nás zatím není tato výuka povinná. Podobně jako v těchto zemích se ale i zde chystá poměrně velká reforma, jejíž cílem je situaci napravit. Není ale snahou ze všech žáků vytvořit programátory, ale jde přesně o schopnost uvažovat více do hloubky a systematičtěji řešit problémy. Záměrem je tedy rozvoj algoritmického myšlení. V porovnání s aktuálním stavem, kdy školství produkuje především uživatele počítačů, neschopné jakékoliv hlubší analýzy problémů, by mělo dojít k diametrálnímu zlepšení. (Bajtler, *Školy informatiku neřeší, říkají zakladatelky kurzů programování pro děti*)

Jedná se ale o výzvu, které stojí v cestě celá řada problémů, od neochoty a především absenci znalostí a zkušeností učitelů v této oblasti až po otázky, kde na tuto výuku najít v limitující časové dotaci potřebný prostor. (Hronová et al., *Děti se budou učit programovat. Novinka ve výuce má být povinná už od první třídy*)

6.4.1 Švédsko

V novém kurikulu je programování integrováno zejména do tří předmětů: technologie, matematika a společenské vědy. Ve společenských vědách se žáci v největší míře zaměřují na sociální aspekt problematiky. Naopak v technologiích se učí mimo jiné ovládat předměty. Začínají jednoduchým programováním, později tvoří vlastní algoritmy a dále je zdokonalují a optimalizují. V neposlední řadě se s programováním setkávají v předmětu matematika, kde se žáci nejprve seznamují s tím, jak se mohou instrukce skládat krok po kroku do větších celků. Výuka tohoto bloku může probíhat s pomocí technologií, ale rovněž jako aktivita unplugged. Později se žáci seznámí s konceptem algoritmus a tím, jak může být vytvářen a použit při programování. V tomto stádiu se učí programovat skrze grafická programovací prostředí (např. Scratch). A nakonec se žáci obeznámí i s dalšími programovacími prostředími a tím, jak v nich mohou být algoritmy sestavovány, testovány a vylepšovány, zejména z pohledu řešení matematických problémů.

Z nového kurikula plyne, že žáci, kteří takto dimenzovanou výukou projdou, budou mít v aspektu algoritmického myšlení své schopnosti na vysoké úrovni. Rovněž budou mít oproti žákům standardních českých škol zásadní náskok. (Balanskat et al., *Strategies to include computational thinking in school curricula*, s. 12-13)

6.4.2 Norsko

Například sousední Norsko není ale ve výuce algoritmického myšlení na tak pokročilé úrovni. Začlenění programování nicméně nově plánuje v přepracovaném kurikulu pro Matematiku a Vědu a to v roce 2020. Již několik let se dokonce pilotuje na nižších stupních u 144 vybraných základních škol a to v rámci volitelného předmětu. (Balanskat et al., *Strategies to include computational thinking in school curricula*, s. 13-15)

6.4.3 Polsko

Příznivěji se k algoritmizaci staví Polsko. Výuka informatiky na rozdíl od českých škol probíhá nepřerušovaně v každém ročníku a to již od první třídy. Součástí polského předmětu informatika je tak rovněž programování a s tím související řešení problémů. Žáci algoritmy navrhují a následně převádí do praxe jejich realizací v programovacím prostředí. Zájmem kurikula přesto není ze všech žáků vytvořit programátory a žáci se tak setkávají pouze s aspekty programování a jejími konstrukty, které jsou potřebné ke splnění cíle. Programování tak žákům slouží především jako nástroj. (Lessner, *Střípky z konference Didinfo 2015 (2): Výuka informatiky v Polsku*)

6.4.4 Velká Británie

I anglické kurikulum se algoritmickým myšlením zabývá. Předmět computing pak dělí na čtyři navazující části, v nichž postupně algoritmické myšlení rozvíjí. Žáci se nejprve seznámí se samotným pojmem algoritmus. Tím, že je přesný a deterministický, a také tím, jak je implementován ve formě programů na digitálních zařízeních. Při tom žáci vytvářejí jednoduché programy a hledají v nich chyby za účelem jejich odstranění. Ve druhém kroku se žáci setkávají s náročnějšími úkoly. Řeší problémy tak, že je rozkládají na menší části. Využívají logického uvažování k tomu, aby odůvodnili, jak jednoduché algoritmy fungují, nebo proč některé nefungují, a jak by je opravili. Následně se žáci seznamují se standardními algoritmy pro řešení některých obecných úkolů, kam spadají například různé typy algoritmů řazení. Své obzory rozšiřují o Booleovskou logiku. Žáci se rovněž seznamují s alespoň dvěma programovacími jazyky, z nichž alespoň jeden musí být textového charakteru. V závěrečné fázi poté dochází k dalšímu rozvoji nabytých zkušeností. (*National curriculum in England: computing programmes of study*)

6.4.5 Obecné požadavky na české školy

Rámcové vzdělávací programy pro gymnázia uvádí jako jeden z cílů vzdělávání potřebu: „uplatňování algoritmického způsobu myšlení při řešení problémových úloh“, jejíž výuka spadá do učiva „algoritmizace úloh – algoritmus, zápis algoritmu, úvod do programování“. (*Rámcovým vzdělávacím programem pro gymnázia*)

Podrobněji se tématu věnuje například RVP pro střední odborné vzdělávání, obor Informační technologie, kde je požadována znalost vlastností algoritmu, schopnost analyzovat úlohu a algoritmizovat ji a schopnost zápisu algoritmu vhodným způsobem, tzn. že je kladen konkrétní požadavek na to, aby žáci „algoritmizovali úlohy a tvořili aplikace v některém vývojovém prostředí“. (*Rámcový vzdělávací program pro obor vzdělání 18-20-M/01 Informační technologie*)

6.4.6 Společná část maturitní zkoušky, informatika

Rovněž dokumentace k původní společné části maturitní zkoušky z informatiky, která měla být platná od školního roku 2011/2012, hovoří o požadavku znalosti algoritmizace a základů programování. Žáci se tedy pro komplexní přípravu na maturitu měli již v této době algoritmizací zabývat.

Platilo to především pro vyšší úroveň společné (státní) části maturitní zkoušky z informatiky: „Řečí tematických celků je v základní úrovni důraz (v mantinelech daných katalogy požadavků) na celky 3, 6, 7 a 8 (operační systém, kancelářské aplikace), kdežto ve vyšší úrovni se přesouvá k celkům 1, 2, 8 a 9 (informatika, hardware, databáze, algoritmizace).“ (*Nejčastější dotazy k informatice*)

Katalog požadavků pro předmět informatika (nižší úroveň) nedefinuje, jak taková na algoritmizaci zaměřená úloha může vypadat, a ani ji nezačleňuje v ukázkovém testu. (*Katalog požadavků zkoušek společné části maturitní zkoušky*)

Naopak katalog pro vyšší úroveň se již k požadavkům blíže rozepisuje a uvádí požadavek na znalost základních programových struktur, jako jsou příkazy, podmínky a cykly. Vyžaduje také schopnost definovat funkci či proceduru a používat matematické, relační a logické operátory. Stejně tak klade potřebu znát pojem proměnná, umět rozpoznat rozsah platnosti proměnné a rozlišovat základní typy hodnot. Pro bližší představu je k dispozici rovněž i ukázková testová úloha, která zapsána nejspíše v pseudokódu požaduje nalezení chyby a označení řádku, ve kterém se tato chyba nachází. Použitá ukázka ale není příliš povedenou úlohou, neboť s lehkým odstupem je možné zjistit, že se chyba nenachází v samotné logice programu, nýbrž pouze ve výpisu hodnoty a to použitím nesprávné, navíc v tom okamžiku nedefinované, proměnné:

Následující program má najít největší hodnotu ze zadaného seznamu přirozených čísel. Program obsahuje chybu. Na kterém řádku je chyba? Vyberte jednu z následujících možností A–D. **A)** na druhém **B)** na čtvrtém **C)** na pátem **D)** **na šestém**

```
1  seznam = [56, 12, 23, 76, 89, 17, 98, 19]
2  nejvetsi = 0
3  for cislo in seznam:
4       if cislo > nejvetsi:
5            nejvetsi = cislo
6  print "Nejvetsi cislo ze seznamu je", cislo
```

Obrázek 1: Ukázková úloha z vyšší úrovně připravované společné části maturitní zkoušky z informatiky.

Katalogy požadavků ke společné části maturitní zkoušky z informatiky již naneštěstí nejsou na webu novamaturita.cz dostupné, a to ani pod odkazy na ně vedoucími[6]. Pro jejich získání tak bylo nezbytné kontaktovat přímo Cermat.

Zkouška z informatiky nicméně nebyla nikdy plošně testována a zůstala pouze ve stádiu plánování. Maturita z informatiky je tak výhradně součástí profilové části (školní) maturity.

Protože české školy zatím většinově výuku algoritmického myšlení podceňují, stojí za povšimnutí alespoň několik málo příkladů, kde tento aspekt rozvoje lidské mysli berou vážně a předčí požadavky RVP. Níže uvedené dva příklady středních škol byly vybrány jednak na základě vlastní zkušenosti a tím i povědomí o výuce algoritmizace na zmíněné škole, tak i na základě doporučení vedoucí diplomové práce.

6.4.7 Smíchovská střední průmyslová škola

Tato pražská střední škola se, na základě slov ředitele pronesených v rámci konference DigiKoalice dne 1. března 2018, snaží držet krok s trendy a aktivně se zapojuje do mnoha projektů. Nejspíše i proto zařazují do výuky předmět Programování, v jehož popise se nachází mj. následující pasáž:

„Cílem je naučit studenty vytvářet algoritmy a pomocí programovacího jazyka napsat zdrojový kód programu.

Absolvent zná vlastnosti algoritmů, umí analyzovat úlohu a algoritmizovat ji, zapsat algoritmus vhodným způsobem. Je schopen používat základní datové typy, řídící strukturu programu, vytvářet jednoduché strukturované programy.“

Žáci si dále mohou zapsat volitelný předmět, který jde ještě dále a snaží se je přimět nad programy přemýšlet hlouběji a hledat lepší řešení:

„Absolvent je schopen prakticky využívat získané znalosti a vytvářet optimální programy z hlediska funkce, rychlosti a velikosti vytvářeného programu pro internetové prostředí a pro tvorbu a správu databází.“

Není tedy překvapivé, že se místní žáci setkají s mnoha zařízeními i nástroji, s jejichž pomocí se postupně propracovávají ke složitějším programům. Nejlepší z nich se pak účastní různých soutěží

[6] *Tisková zpráva ke zveřejnění ilustračního testu ze zkušebního předmětu informatika* [online]. Praha: CERMAT (Centrum pro zjišťování výsledků vzdělávání), c2010 [cit. 2019-04-04]. Dostupné z: https://www.novamaturita.cz/tiskova-zprava-ke-zverejneni-ilustracniho-testu-ze-zkusebniho-predmetu-informatika-1404035852.html.

nebo již pracují jako programátoři. Rovněž firmy se o místní žáky zajímají a nabízejí jim již v průběhu studia či po jeho skončení zajímavé pracovní pozice. (*Profil absolventa - IT*)

6.4.8 Obchodní akademie, Střední odborná škola knihovnická a Vyšší odborná škola Brno, příspěvková organizace

Další institucí, která se staví k informačním technologiím a především k aktuálním výukovým trendům čelem, je tato brněnská střední škola. Na základě jejich ŠVP (školní vzdělávací program)[7] je zjevné, že se škola nevyhýbá algoritmizaci ani nyní a snaží se žáky naučit logickému a algoritmickému myšlení a tvorbě programů v nejrůznějších programovacích jazycích:

„Učivo předmětu programování vede žáky nejprve k algoritmizaci jednoduchých problémů, později se obtížnost úloh zvyšuje. Žáci se seznámí se základními principy algoritmizace úloh, zanalyzují zadaný problém a zapíší vytvořený algoritmus způsobem vhodným pro danou konkrétní úlohu. Žáci se seznámí s postupy při návrhu SW, základy UML jazyka a Use Case diagramy. Žáci se seznámí s principy strukturovaného programování. Ve svých programech používají základní datové typy i potřebné příkazy programovacího jazyka. V procesu ladění žáci odstraní veškeré chyby v programu tak, aby výsledná aplikace pracovala podle zadání. Žáci se seznámí také s principy objektového programování. Porozumí pojmům třída, objekt a vyjmenují jejich základní vlastnosti. V programech aktivně použijí jednoduché objekty a události. Žáci se také podrobněji seznámí s problematikou tvorby WWW stránek včetně použití databází ve webovém programování. Podstatnou část předmětu představuje samostatná tvorba jednoduchých aplikací včetně statických a dynamických WWW stránek Pro řešení běžných konkrétních úkolů žáci vyberou vhodné programové vybavení a za pomoci manuálu a nápovědy se dokáží sami seznámit s novými aplikacemi.

Všechny výše uvedené kompetence umožňují žákům začlenění do moderní informační společnosti.“

Z výše uvedené citace z ŠVP pro Informační technologie je zjevné, že se žáci seznámí z celou řadou postupů a nástrojů a získají široký rozhled. Stejné náplně se dočkají také žáci oboru Kybernetická bezpečnost. Trochu odlišného pojetí se dočkají žáci oboru Telekomunikace, ovšem samotná algoritmizace se ve všech třech případech vyučuje stejným způsobem.

ŠVP dále uvádí tyto cíle výuky algoritmizace:

- „ovládá principy algoritmizace úloh a sestavuje algoritmy řešení konkrétních úloh
- zná vlastnosti algoritmu
- zanalyzuje úlohu a algoritmizuje ji
- zapíše algoritmus vhodným způsobem“

[7] *Školní vzdělávací program* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/%C5%A0koln%C3%AD_vzd%C4%9Bl%C3%A1vac%C3%AD_program; *Rámcové vzdělávací programy* [online]. Praha: Národní ústav pro vzdělávání [cit. 2019-04-04]. Dostupné z: http://www.nuv.cz/t/rvp.

6.4.9 Další oslovené školy

Pro bližší a především aktuální přehled o stavu výuky algoritmického myšlení na českém území došlo během výzkumu také k oslovení více než stovky středních škol, především pak (avšak nikoliv pouze) obecně zaměřených gymnázií rozmístěných po celé České republice. Kromě gymnázií byly kontaktovány rovněž střední odborné školy se zaměřením na IT. Snahou bylo kontaktovat vždy několik škol v různých krajích, přičemž k tomuto účelu byl využit seznam škol dostupný na adrese seznamskol.eu.

Školám byla kladena otázka, zda využívají při výuce informatiky metody či nástroje pro rozvoj algoritmického myšlení a pokud ano, v jakém ročníku se s ním žáci setkávají a o jaké prostředky se jedná.

Tabulka, která je k nahlédnutí v přílohách této práce na straně 117, uvádí většinu škol, které na dotaz zareagovaly, vyjma několika málo případů, v nichž byla odpověď školy vzhledem k výuce programování negativní. Může však docházet k určitému zkreslení, neboť značné procento kontaktovaných škol vůbec na položený dotaz neodpovědělo. Není možné vyloučit, že to mohlo být právě proto, že výuce algoritmického myšlení není na těchto školách kladen dostatečný důraz. Zvláště o tom pak vypovídají průzkumy, které hovoří o nízké kvalitě výuky informatiky na našich školách. Může se jednat o nedostatek kompetencí učitele k výuce informatiky či o nezájem vedení školy investovat do výuky informačních technologií větší množství času. Situaci nepomáhá ani skutečnost, že v současném a stále platném RVP je náplní práce učitele příslušného předmětu učit žáky nejen pracovat s počítačem, ale i ovládat a pracovat s celou řadu programů, jež svými možnostmi daleko více zapadají do jiných předmětů, jako je matematika či český jazyk. Záleží pak na každé škole případně i učiteli informačních technologií, jak se k k výuce postaví a zda bude nad rámec požadavků učit i algoritmizaci a programování. V současnosti však tato výuka není celoplošná a množství žáků se s ní tak, navzdory požadavkům dnešní doby, které řadí programování mezi kruciální znalosti, na školách vůbec nesetkává. (*Ne jen uživatelé, ale programátoři. Výuku informatiky mají změnit nové učebnice*; "Kvalita a efektivita vzdělávání a vzdělávací soustavy ve školním roce 2017/2018", s. 237)

Ze získaných informací je na druhou stranu ale patrné, že problematika výuky algoritmizace je na některých školách již v zájmu pedagogů a je i v rámci povinných hodin informatiky začleňována. Jsou však viditelné poměrně výrazné rozdíly v tom, jaké nástroje a technologie k výuce různé školy používají a především pak do jaké míry se problematikou v hodinách zabývají.

6.4.10 Informatické soutěže

Kromě standardní náplně hodin se mohou žáci nebo i školy zapojovat do celé řady informatických soutěží. Velice známý je například algoritmizaci obecně pojímající Bobřík informatiky[8], který pokrývá taktéž středoškoláky, včetně nejvyšších ročníků. Těm na nižším stupni (prima až kvarta) se pak věnuje také soutěž KoduCup[9], zaměřující se na práci v interaktivním 3D prostředí Kodu. Žáci středních škol mohou uplatnit, ale i rozšířit, své dovednosti ve světe Minecraft (edice Minecraft Education) také v soutěži Minecraft Cup[10].

Ačkoliv jsou informatické soutěže vhodné pro rozvoj znalostí žáků, samotný jejich úspěch v nich nemusí mít tak zásadní význam, jak by se mohlo zdát:

„Přiznejme si, že programátorské soutěže jsou na tom obdobně jako testy inteligence. Ty také neměří nic jiného, než schopnost řešení testů inteligence. Všichni víme, že korelace naměřeného IQ s úspěšností v praktickém životě není tak velká, jakou by ji rádi viděli autoři IQ testů. Absolventi, kteří dokáží skvěle řešit různé algoritmické lahůdky na soutěžích, mohou mít velké problémy v praxi, kde je zaměstnavatel postaví před obludný program vytvořený jejich předchůdci a zadá jim úkol doplnit program nějakou další funkcí nebo odladit nějakou jeho problematickou pasáž.“ (Pecinovský, *Metodika výuky programování na rozcestí*)

Na druhou stranu tyto soutěže typicky pokrývají obecná témata, jejichž znalost se účastníkům může hodit nejen v pozdějším studiu, ale i při vykonávání jejich profese. Rovněž mohou mít pozitivní dopad na vlastní zájem žáků o ICT, kteří se s jejich pomocí mohou sami začít problematikou zabývat a nadále své schopnosti rozvíjet.

V českém prostředí rovněž vznikla organizace sdružující pedagogy se změřením na ICT. Jejich cílem je aktivně se podílet na zavádění ICT do výuky a získávat a šířit zkušenosti s jejich využíváním ve škole.

[8] *Bobřík informatiky* [online]. České Budějovice: Pedagogická fakulta Jihočeské univerzity v Českých Budějovicích, c2008-2018 [cit. 2019-04-04]. Dostupné z: https://www.ibobr.cz.

[9] *Kodu Cupu Česko* [online]. Pardubice, Praha: DELTA - Střední škola informatiky a ekonomie Pardubice a Vzdělávací centrum Microsoft ZŠ a MŠ Červený vrch Praha 6, 2019 [cit. 2019-04-04]. Dostupné z: http://www.koducup.cz.

[10] *Minecraft Cup* [online]. Praha: Microsoft, c2019 [cit. 2019-04-04]. Dostupné z: http://minecraftcup.cz.

7 Možné prostředky pro výuku algoritmizace

Výuka programovaní by měla být pro žáky co nejzáživnější a jejich tvorba, navzdory tomu, co čeká typického programátora, co nejhmatatelnější tak, aby byly jednotlivé části kódu co nejlépe uchopitelné a bylo jednoznačné, k čemu jsou potřeba a co dělají. Vytvořené programy by tak měly být ideálně interaktivní, reagovat na uživatele a vracet nějaký výstup.

K dosažení výše zmíněných cílů je možné využít celé řady prostředků. Školy a jejich učitelé mají široké možnosti[11] v nákupu specifických programovatelných zařízení vhodných či přímo určených pro výuku. (*Z výzev OP VVV na šablony je možné pořídit i didaktické pomůcky a IT techniku*) Zde spadají zejména roboti. Nebo se mohou spolehnout pouze na počítač a software v něm a programování pojmout jako tvorbu desktopových aplikací.

Samostatné robotické jednotky (Například programovací robot Edison)[12] a zvláště robotické stavebnice, jako jsou LEGO[13] či Merkur[14], poskytují rozmanité možnosti jejich užití, neboť je možné je sestavit do různých tvarů, vybavit volitelnými součástkami a naprogramovat k mnoha odlišným činnostem. Práce s nimi je pro žáky typicky záživná, protože mohou modifikovat nejen software, ale upravují si i hardware. Roboti žákům rovněž poskytují ze všech možných metod nejvíce hmatatelné výsledky, protože mohou zasahovat do reálného světa – pohybovat se, uchopovat předměty atp.

V pokročilejší úrovni studia se pak jako vhodné alternativy robotů jeví mikropočítače, typicky Arduino[15], které o sobě hovoří jako o jednom z nejúspěšnějších nástrojů pro koncept výuky STEM/S-TEAM[16](*About Us*), nebo výkonnější a plnohodnotný počítač Raspberry Pi[17]. Protože se v základu jedná o jednoduché destičky bez jediného viditelného výstupu (pomineme-li zabudovanou led diodu), je potřeba je doplnit dalšími prvky. Typicky se k tomu využívají samostatné součástky, které se připojují do nepájivého pole a při správném propojení je možné je různě ovládat skrze náš program. Tak si žáci mohou připojit například led diody, motory, relé i displeje či naopak různá čidla, např. teploty, vlhkosti či vzdálenosti. Možnosti jsou v tomto směru obrovské a mohou vést třeba i k sestavení si vlastního robota zcela od základů. Na druhou stranu tato práce vyžaduje již pokročilé znalosti jak hardwaru, tak softwaru.

[11] *Operační program Výzkum, vývoj a vzdělávání* [online]. Praha: Ministerstvo školství, mládeže a tělovýchovy, c2017 [cit. 2019-04-04]. Dostupné z: https://opvvv.msmt.cz.

[12] *Osobní robot* [online]. Zeleneč: Profess Consulting s.r.o., c2019 [cit. 2019-04-04]. Dostupné z: http://osobnirobot.cz.

[13] *Mindstorms LEGO.com* [online]. Billund (Dánsko): LEGO Group, c2019 [cit. 2019-04-04]. Dostupné z: https://www.lego.com/cs-cz/mindstorms.

[14] *Merkurtoys* [online]. Police nad Metují: Merkurtoys s.r.o., c2016-2019 [cit. 2019-04-04]. Dostupné z: http://www.merkurtoys.cz.

[15] *Arduino* [online]. Somerville, Massachusetts (USA): Arduino LLC, c2019 [cit. 2019-04-04]. Dostupné z: https://www.arduino.cc.

[16] *Koncept STEM* [online]. Praha: Národní ústav pro vzdělávání [cit. 2019-04-04]. Dostupné z: http://www.nuv.cz/p-kap/koncept-stem.

[17] *Raspberry Pi* [online]. Cambridge: Raspberry Pi Foundation, [2019] [cit. 2019-04-04]. Dostupné z: https://www.raspberrypi.org.

Je ale možné se tomu vyhnout, například s využitím některého z kompletně sestavených výukových „shieldů“, tedy destiček, které přímo pasují ke konkrétnímu zařízení a bývají typicky osazeny celou řadou již připojených vstupních a výstupních prvků. Ty bývají zdrojem jak vizuálních, tak i zvukových podnětů, mívají na sobě tlačítka, displej i piny pro další možná rozšíření. Hotovou destičku stačí k mikropočítači pouze připojit tak, jak je, a poté se již zabývat pouze softwarovou stránkou.

Výhodou Arduina a Raspberry Pi je, že se jedná o poměrně levnou záležitost, která je velmi univerzální a poskytuje prostor pro manipulaci s hardwarem, jíž je ale možné se do velké míry vyhnout, či naopak, mají-li žáci zájem, je možné ji neplánovaně a snadno do výuky i později v průběhu zařadit. (Davis, *15+ Ways of Teaching Every Student to Code (Even Without a Computer)*)

Ačkoliv mají roboti i mikropočítače bezesporu celou řadu předností, mají také svá negativa. Jednak je nezbytné, aby do nich škola investovala určité prostředky, čemuž nemusí být vedení školy nakloněno. Je také možné, že práce s hardwarem, i kdyby pouze na jednoduché úrovni, odradí nejednoho učitele informatiky. A v neposlední řadě mají projekty vázané na majetek školy negativní dopad v případě, že by si děti rády své projekty odnesly domů, pochlubily se jimi rodičům či na nich dále sami ve volném čase pracovali. Samozřejmě by bylo v případě velkého zájmu možné, aby škola zařízení na krátký čas zapůjčila, ovšem ani to nemusí být dostatečné, neboť vyhlídka na potřebu vrátit zařízení může být demotivující. Nákup stavebnice se naopak nemusí vejít do rodinného rozpočtu nebo jen rodičům investice do něčeho takového nepřijde opodstatněná.

Možné řešení se může skrývat například ve využití mobilních zařízení, ideálně těch, co již žáci vlastní a nosí do školy. Zvláště pak na středních školách se jedná o téměř bezproblémový způsob výuky programování, neboť vhodné mobilní zařízení vlastní téměř každý žák.

Mobilní zařízení poskytují dostatečné množství interaktivity, a i když zaostávají v možnostech rozšiřování a úprav hardwarových periferií, mají na druhou stranu velké přednosti v síťové konektivitě a vizuálních možnostech, neboť spoustu prvků mohou vykreslit virtuálně přímo na obrazovce. Ani vestavěné funkce telefonu nejsou zanedbatelné, protože je možné využívat mnoha různých senzorů.

Díky tomu škola nemusí investovat tak velké finanční prostředky k realizaci a ani žáci nejsou zásadně ochuzeni, neboť se na druhou stranu seznámí s prvky, které by byly v případě robotů a mikropočítačů obtížněji dostupné.

Tabulka 1: Srovnání charakteristických vlastností jednotlivých prostředků

	Desktopový počítač	**Roboti**	**Robotické stavebnice**	**Mikro-procesory (Arduino)**	**Mobilní zařízení**
Okamžitý náhled (bez nutnosti nahrávat software)	Lze	Ne	Ne	Ne	Ne
Grafické rozhraní	Lze	Typicky ne	Typicky ne	Samo o sobě ne	Ano
Podpora většího množství programovacích jazyků	Ano	Může být problém	Může být problém	Může být problém	Ano
Hardwarová flexibilita (doplňky)	Např. skrze USB, nicméně pro přístup k zařízení je potřeba vytvořit komplikovanější program.	Typicky ne	Ano	Ano	Např. skrze USB, nicméně pro přístup k zařízení je potřeba vytvořit komplikovanější program. Přístup navíc nemusí OS vůbec dovolit.
Zabudované senzory	Jen velmi málo	V závislosti na typu robota	Ne, lze připojit	Ne, lze připojit	Typicky velké množství
Přenositelnost	Pokud se nejedná o notebook, ne	Ano	Ano	Ano	Ano
„Hmatatelnost“ výsledků, názornost	Obrazovka, web, reproduktory	Samotný pohyb robota a jeho interakce s prostředím	Samotný pohyb robota a jeho interakce s prostředím	Připojený displej, web	Výstupy aplikace se zobrazují na displeji

* Výše uvedená tabulka zobrazuje typická užití. Nevylučuje různá jiná zapojení, díky kterým je možné například využít i klasický stolní počítač podobně, jako např. Arduino nebo robota.

8 Programovací jazyky

Programovací jazyky umožňují předávání instrukcí počítači ve formě, které počítač rozumí. Podobně jako je tomu i s přirozenými jazyky, existuje také množství programovacích jazyků. Programy napsané v programovacích jazycích ale nejsou procesorem přímo vykonávány. V některých případech je nutné je (ať již před spuštěním či za chodu) překládat až na úroveň binárního kódu. Děje se tak s pomocí překladačů, přičemž se tento proces označuje za kompilaci. U interpretovaných jazyků se pak o chod programu stará příslušný interpret.

Programovací jazyky se liší některými svými vlastnostmi, jako jsou definice stavů, výchozí hodnoty atp., ale v závislosti na rodině jazyků víceméně i v syntaxi. Nicméně způsoby sestavování kódu jsou u všech běžných imperativních programovacích jazyků velmi blízké a učení se novým jazykům proto zpravidla nebývá obtížné. (*What are Computer Programming Languages?*)

V závislosti na tom, pro jaké zařízení je program uvažován, je na výběr z menší nebo větší škály programovacích jazyků.

Při zaměření se na roboty, například LEGO® MINDSTORMS®, je vývojář obvykle odkázán na výrobce, který spolu s hardwarem dodává i vývojové prostředí. V případě produktů určených pro školy to potom typicky znamená výuku ve výrobcem doporučeném prostředí i jazyce. Nicméně většinu těchto zařízení je možné programovat také pomocí nástrojů třetích stran, jejichž hlavním smyslem je právě skutečnost, že nabízejí i další jazyky, například hojně rozšířený Python[18], Javu[19] nebo C[20]/C++[21].

Jestliže zvolí vývojář jako cílovou platformu Arduino či jeho klony, může využívat jazyky C/C++. Totéž platí i o WiFi modulu ESP8266[22], kde je kromě kompilovaných jazyků možné využít také interpretovaný jazyk LUA[23], je-li využit alternativní firmware NodeMCU[24]. (*NodeMCU Documentation*)

Také mobilní zařízení (tedy mobilní telefony a tablety) mají svá omezení a v případě Androidu je vývojář typicky odkázán na Javu, i když psaní kódu v jazycích, jako jsou C/C++, Python, ale i mnoho dalších, je možné.

Zatím nezmíněnou možností je oblast webu a tvorby webových stránek a aplikací. Zde se typicky preferuje JavaScript na straně klienta a PHP[25] na straně serveru, ačkoliv je v posledních letech tato technologie mírně na ústupu a i na straně serveru se v určitých případech stále více prosazuje

[18] *Python* [online]. Wilmington, Delaware (USA): Python Software Foundation, c2001-2019 [cit. 2019-04-04]. Dostupné z: https://www.python.org.

[19] *Java* [online]. Redwood City, Kalifornie (USA): Oracle, 2018 [cit. 2019-04-04]. Dostupné z: https://www.java.com.

[20] *C (programming language)* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-04-04]. Dostupné z: https://en.wikipedia.org/wiki/C_(programming_language).

[21] *C++* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-04-04]. Dostupné z: https://en.wikipedia.org/wiki/C++.

[22] *ESP8266* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-04-04]. Dostupné z: https://en.wikipedia.org/wiki/ESP8266.

[23] *The Programming Language Lua* [online]. Rio de Janeiro (Brazílie): LabLua, 2018 [cit. 2019-04-04]. Dostupné z: https://www.lua.org.

[24] *NodeMcu* [online]. NodeMcu Team, c2014-2018 [cit. 2019-04-04]. Dostupné z: https://www.nodemcu.com.

[25] *PHP: Hypertext Preprocessor* [online]. PHP Development Team, Zend Technologies, c2001-2019 [cit. 2019-04-04]. Dostupné z: https://www.php.net/.

JavaScript[26] (v podobě běhového prostředí[27] Node.js[28]). (Oram, *Is PHP relevant?*)

Největší možnosti se naskytují, vyvíjí-li autor software pro osobní počítače. V takovém případě má nad softwarem a jeho spouštěním největší kontrolu a může ho, především díky menším omezením na straně hardwaru, vyvíjet takřka v libovolném programovacím jazyce.

8.1 Dělení programovacích jazyků

Programovací jazyky lze dělit na základě míry abstrakce či způsobu překladu. Vyšší programovací jazyky dále pak na imperativní a deklarativní jazyky. (*Programovací jazyk*)

8.1.1 Podle míry abstrakce

1. **Nižší programovací jazyky**
   Zde spadá jazyk symbolických adres, který symbolicky reprezentuje jednotlivé strojové instrukce a konstanty, což je nezbytné pro tvorbu strojového kódu pro konkrétní procesor. Znamená to tedy, že kód napsaný v tomto jazyce je vázaný na konkrétní typ procesoru, popř. rodinu procesorů a je obtížně přenositelný na jiné platformy.
   Tento jazyk se dále překládá do přímo spustitelného strojového kódu pomocí assembleru.
   Programovat v jazyce symbolických adres je možné například pro Arduino[29], Raspberry Pi[30] nebo i běžný počítač. Vzhledem k náročnosti tohoto způsobu, tj. složitosti zápisu téměř libovolného algoritmu, navíc vyžadující znalost jednotlivých instrukcí zvoleného typu procesoru, není vhodné začínat výuku v tomto jazyce.
2. **Vyšší programovací jazyky**
   Do této kategorie se řadí všechny ostatní jazyky, které přinášení vyšší míru abstrakce a tím se stávají také univerzální a nezávislé na procesoru, ne kterém běží. Kromě abstrakce nad konkrétními instrukcemi procesoru se také mnohem více blíží tomu, jak lidé přemýšlejí, využívají klíčových slov z přirozeného jazyka a usnadňují a zkracují zápis mnoha činností, které by se v nižším jazyce psaly zdlouhavěji.

8.1.2 Podle způsobu překladu

1. **Kompilované jazyky**
   Jazyky, které je nutno před jejich spuštěním překládat do strojového kódu, čímž nám vznikne na zařízení binární soubor, spadají mezi jazyky kompilované. Jejich výhodou je rychlejší běh, avšak na druhou stranu je také potřeba kompilace pro různé architektury v případě, že je cílem spouštět program na různých zařízeních. Spadají zde například jazyky C a C++.

[26] *JavaScript* [online]. Mountain View, Kalifornie (USA): Mozilla Corporation, 2019 [cit. 2019-04-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[27] *Běhové prostředí* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2017 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/B%C4%9Bhov%C3%A9_prost%C5%99ed%C3%AD.

[28] *Node.js* [online]. San Francisco (CA): OpenJS Foundation a The Linux Foundation, [2018] [cit. 2019-04-04]. Dostupné z: https://nodejs.org.

[29] *Programming Arduino Uno (ATmega386P) in assembly* [online]. San Francisco (CA): GitHub, 2015 [cit. 2019-04-04]. Dostupné z: https://gist.github.com/mhitza/8a4608f4dfdec20d3879.

[30] Thiebaut, Dominique. *Tutorial: Assembly Language with the Raspberry Pi* [online]. Northampton (Anglie): Clark Science Center Smith College, 2015 [cit. 2019-04-04]. Dostupné z: http://www.science.smith.edu/dftwiki/index.php/Tutorial:_Assembly_Language_with_the_Raspberry_Pi.

2. **Interpretované jazyky**
   Druhá kategorie obsahuje jazyky, které není nutno před jejich spuštěním překládat. Tyto spoléhají na interpret, tedy nástroj, který umožňuje spouštění takto napsaných programů. Typickým příkladem je například shell[31], který umožňuje běh skriptů, nebo serverové PHP, ale i JavaScript, Ruby, Python a Java. V dnešní době se většina interpretovaných jazyků nekompiluje řádek po řádku za běhu programu, ale používá tzv. mezikód. Výsledkem je kombinace interpretovaného a kompilovaného přístupu, kdy je zdrojový kód překladačem nejprve přeložen např. do bytekódu. Ten je snadno interpretovatelný příslušným virtuálním strojem[32] a vyniká zejména platformní nezávislostí. Program může být do mezikódu překládán při každém spuštění (Ruby), při detekci změny zdrojového kódu (Python), nebo jen jednorázově (Java). Java zachází ještě dál a v kombinaci se systémem JIT (překlad Just In Time) přidává i podporu kompilace do strojového kódu, která se provádí za běhu programu, čímž ještě více optimalizuje a tím pádem i zrychluje běh výsledné aplikace. (*Oracle JVM Just-in-Time Compiler (JIT)*)

8.1.3 Dělení vyšších programovacích jazyků

1. **Procedurální (imperativní) jazyky**
   Nejčastěji se vývojáři při programování setkávají s jazyky, které spadají mezi tzv. imperativní jazyky. To jsou jazyky, kde programátor popisuje cestu k řešení pomocí posloupnosti příkazů a určuje přesný postup – vytváří tedy algoritmus. Program sestává ze sady proměnných, které mění na základě vyhodnocování podmínek svůj stav. (*Imperativní programování*)
   (a) **Strukturované** – Algoritmus se dělí na jednotlivé podúlohy a souhrnně tvoří celek.
   (b) **Objektově orientované** – Program se skládá z objektů, které zapouzdřují jednotlivé proměnné a metody, mohou se dědit a lze s nimi pracovat, aniž by do nich bylo z vnější vidět, tj. mohou fungovat jako černé skříňky a komunikovat na základě daného rozhraní.
2. **Neprocedurální (deklarativní) jazyky**
   Psaní programu v deklarativním jazyce spočívá v deklaraci toho co se má provést, nikoliv jak se to má provést – jsou tedy opakem imperativního programování. Typickým příkladem jsou databázové jazyky, například SQL. Dále třeba Scheme nebo Prolog. (*Deklarativní programování*)

8.2 Možné způsoby tvorby programů

Ať už je zvolenou cílovou platformou cokoliv, tj. pouze počítač, mobilní zařízení či robot, vždy je v případě vývoje softwaru nezbytné zvolit vhodné programovací prostředí. Zvláště pak ve školním prostředí učitel zvolí a doporučí jeden konkrétní nástroj či postup, jež bude představovat všem žákům.

[31] *Shell (informatika)*. San Francisco (CA): Wikimedia Foundation, 2001-2017. Dostupné také z: https://cs.wikipedia.org/wiki/Shell_(informatika).

[32] „Virtuální stroj je v informatice software, který vytváří virtualizované prostředí mezi platformou počítače a operačním systémem, ve kterém koncový uživatel může provozovat software na abstraktním stroji.„ (*Virtuální stroj*)

8.2.1 Textové prostředí

Nejčastějším způsobem tvorby programu je psaní kódu v čistě textovém prostředí. K tomu je možné využít jak klasický textový editor (ideálně alespoň se zvýrazňováním syntaxe, tedy takový, který psanému jazyku rozumí), tak rovnou celé vývojové prostředí.

Integrované vývojové prostředí (IDE) má oproti prostému textovému editoru celou řadu předností, neboť na rozdíl od něj nejenže zvýrazňuje syntax, ale umožňuje také správu celého projektu, běžně i s dalšími soubory, které k danému programu patří. Tj. je si vědom také multimediálních souborů či dokumentace, která se v případě většího projektu typicky vytváří jako jeho součást. Vývojová prostředí zahrnují či komunikují také s překladačem, je-li potřeba, a umožňují rychlé sestavování projektu pro jeho náhled i finální sestavení. Pokud jsou IDE využívána nejčastěji s jazykem, kterému rozumí (v opačném případě ztrácí jeho užití význam), podporují také doplňování rozepsaných klíčových slov, jsou schopny zobrazit nápovědu pro to, co která knihovní funkce znamená, a propojují jednotlivá volání funkcí a metod tak, že mají povědomí o tom, odkud se volají, a jsou schopny na ně v případě potřeby rychle odkázat – jsou-li pominuty extrémní případy, kdy kvůli složitosti psaného kódu nefungují některé nápovědy správně. Kromě toho se dokáží často starat také o verzování a podporují i celou řadu pokročilejších akcí souvisejících se správou projektu a jeho sestavování.

Spadají zde všechny výše uvedené programovací jazyky, tedy C, C++, Ruby, Python, Java, JavaScript, Lua, shellové skripty či SQL.

Zásadní překážkou ve školním prostředí ale může být skutečnost, že tento způsob programování vždy vyžaduje poměrně velkou znalost jazyka k tomu, aby v něm byla práce pro žáky pohodlná, rozuměli tomu, co píší, a dokázali vhodně reagovat na nové požadavky na rozšiřování aplikace. Rovněž je velice citlivé na překlepy, i když v tomto ohledu může nápověda editoru výrazně pomoci.

8.2.2 Grafické (blokové) prostředí

Pro začátečníky, zvláště pak ve školách, kde není primárním cílem žáky naučit pracovat s konkrétním programovacím jazykem, ale spíše je pouze seznámit s algoritmizací jako takovou, se tak jako vhodnější jeví spíše blokové editory. Ty sestávají z jednotlivých do sebe zapadajících bloků a podobně jako puzzle je možné zanořovat jen bloky, které do sebe pasují.

Tato prostředí vypadají většinou velmi podobně bez ohledu na to, jaké zařízení mají ovládat. Díky tomu by neměl být zásadní problém ani v případě, že bude učitel s žáky prozkoumávat více možností, neboť různé blokové editory mohou pomoci vytvořit program jak pro počítač, tak pro Arduino i mobilní telefony. Zároveň vždy oddělují programátora od samotného kódu v textové podobě a nevyžadují tak jeho znalost, což může jednak žákům usnadnit osvojení si tvorby aplikací a především tuto činnost zatraktivnit, neboť se najednou z prostého psaní stává skládačka v barevném interaktivním prostředí. To může mít dopad i na samotnou práci ve skupinkách, neboť všichni žáci takovému „zápisu“ budou rozumět a mohou nad skládačkou mnohem lépe spolupracovat. (Chumpia, *Visual-based vs. Text-based programming languages*)

Mezi vizuální programovací jazyky patří například Scratch či Kodu (grafická nadstavba nad jazykem Logo). (*Visual programming language*) Na tomto principu je postavena celá řada vývojových

prostředí a programovacích her, které jsou často implementovány za pomocí svobodně dostupné knihovny s otevřeným zdrojovým kódem Blockly. (*Blockly*)

9 Tvorba mobilních aplikací

Vývoj mobilních aplikacích je v porovnání s prací na aplikacích pro běžné osobní počítače v mnoha ohledech specifický, ačkoliv to programátora, zvláště při využití některého z pokročilejších vývojových prostředí, které ho od těchto skutečností odstíní, nemusí příliš trápit.

Prvním a asi největším, i když nejméně viditelným rozdílem, je typicky odlišná architektura procesoru mobilního zařízení od procesoru desktopového počítače. Podobně je na tom i většina robotů a mikropočítačů. (*CPU Architecture*) V případě, že autor využije jazyk vyžadující kompilaci do binárního kódu, je potřeba spouštět překladač takovým způsobem, že bude produkovat strojové instrukce srozumitelné pro cílovou architekturu.

K dosažení tohoto cíle existuje několik metod:

1. Nejčastěji se využívá křížových překladů (například pomocí GCC), ideálně v kombinaci s kontejnery k oddělení celé škály využívaných nástrojů od hostitelského systému. (*App development*)
2. Druhou a spíše hypotetickou možností je spouštění kompilace v emulátoru, který umožňuje běh celého operačního systému ve virtualizovaném prostředí, přičemž zároveň překládá instrukční sady tak, že je hostovaný systém stejné architektury jako zařízení, pro které program je program kompilován. Nevýhodou ovšem je potřeba spouštět kompletní operační systém, navíc s hardwarovou emulací, která velmi významně redukuje výkon.
3. Poslední a nejméně vhodnou cestou je kompilace přímo na cílovém zařízení. Zvláště v případě mobilních zařízení zde narážíme na limit kapacity pamětí flash i RAM, stejně jako méně výkonné procesory – ačkoliv všechny tyto aspekty se v průběhu posledních let díky stále výkonnějšímu hardwaru dramaticky zlepšily. Ani tato možnost se v praxi téměř nevyužívá. (Jacewicz, *Geany on Ubuntu Touch device as text editor, source code editor, debugger and compiler for multiple languages*)

Druhým aspektem, v němž se mobilní zařízení liší, je v podstatě nutnost využívat grafické rozhraní. (*Mobile app development*) Ne, že by na těchto zařízeních nebylo možné vytvářet čistě textové či na pozadí běžící aplikace, ovšem z žákovského pohledu se očekává, že bude uživatel s aplikací interagovat pomocí obrazovky a dotyků prsty.

Ať již je cílovou platformou kterýkoliv mobilní systém, je vhodné si dále vhodně zvolit jeden z mnoha dostupných frameworků pro tvorbu grafických aplikací a to zejména na základně potřeb, tedy toho, co bude aplikace dělat a jaké periferie obsluhovat, stejně jako na základě náročnosti implementace aplikace v různých prostředích. (*Comparison of Mobile App Frameworks for Cross-Platform Development in 2019*)

9.1 Mobilní operační systémy

Vývoj aplikací pro mobilní telefony s sebou bohužel přináší také komplikace, zvláště je-li cílem podporovat různé operační systémy. Tzn. soustředí-li autor své síly na tvorbu aplikace pro Android,

může a typicky je problém ji spustit například na iOS a naopak. Existují ale frameworky, které si s touto překážkou dokáží poradit a jsou schopny finální aplikaci zkompilovat a zabalit tak, aby fungovala alespoň na obou nejrozšířenějších mobilních platformách.

Ty se nejčastěji s multiplatformní podporou vypořádávají tak, že obsahují nějaký aplikační základ, postavený specificky pro podporované platformy, který si každý v daném frameworku napsaná aplikace přibaluje, a dále již vykreslují prostředí do okna podobného webovému prohlížeči, tedy stejně jako klasickou webovou aplikaci. Typickým příkladem je například velmi rozšířená Cordova či React Native. Zacházejí ale dále v tom, že umožňují z kódu vnitřní aplikace, nejčastěji běžící v JavaScriptu, obsluhovat i systémové služby a získávat údaje ze senzorů zařízení.

Na druhou stranu existují ale i frameworky, které umožňují spouštění aplikací s klasickým rozhraním na různých systémech. Mezi ty nejrozšířenější a zároveň nejuniverzálnější z této kategorie patří se svou podporou mnoha různých desktopových operačních systémů, iOS, Androidu a dalších bezesporu Qt. (*Comparison of Mobile App Frameworks for Cross-Platform Development in 2019*)

9.1.1 Android

Google Android je nejrozšířenějším mobilním operačním systémem, který nalézá útočiště nejen na mobilních telefonech a tabletech, ale i celé řadě dalších zařízení, jako jsou televize, ale i různé palubní počítače a vestavěné systémy. Díky obrovskému zájmu vývojářů, jak ze strany podpory softwaru, tak co se týče portování systému na další zařízení, je možné Android nainstalovat v různých neoficiálních a různě upravených verzích nejen na telefony a tablety, ale třeba i vedle či místo stávajícího operačního systému na osobní počítač.

Vzhledem k rozšířenosti a i současné otevřenosti samotného systému je Android ideální volbou nejen pro začínající vývojáře, neboť je mu k dispozici nepřeberné množství různých vývojových nástrojů, stejně jako velmi kvalitní dokumentace i online podpora. (*Android (operating system)*)

9.1.2 iOS

Druhým nejrozšířenějším mobilním operačním systémem je Apple iOS, který ale doplácí na svou uzavřenost. Většímu rozšíření brání jednak skutečnost, že je k dispozici pouze na zařízeních prodávaných s tímto systémem – vyžaduje tedy zařízení Apple. Druhým problémem je absence některých možností, které Android podporuje. To svazuje vývojáři do určité míry ruce v tom, co všechno může pro konkrétní zařízení vytvořit. Příkladem velmi dobře zachycujícím tento problém je webový prohlížeč Google Chrome, který na iOS nemůže využívat vlastní vykreslovací jádro, ale pouze systémové. Nejedná se tedy o plnokrevný prohlížeč Chrome, ale pouze o alternativní grafickou nadstavbu nad systémovým prohlížečem. V této spojitosti je potřeba zmínit ještě jednu slabinu iOS a to problém s aktualizacemi. Jakmile Apple rozhodne, že dané zařízení již dále nebude dostávat aktualizace, jsou rovněž některé aplikace bez ohledu na vývojáře zamrazeny ve svých původních verzích. Ve zmíněném příkladě s Google Chrome to znamená, že i když Google vydá aktualizaci pro svůj prohlížeč i pro již nepodporovanou verzi iOS, ani tak nedokáže zajistit podporu nejnovějších technologií vykreslovacím jádrem a moderní prvky jsou tak na zařízení nefunkční stejně, jako ve vestavěném prohlížeči. (Kozub, "Návrh JS knihovny pro tvorbu atypických forem didaktických testů", s. 59)

Android žádným z uvedených problémů netrpí (ač jednotlivá zařízení s ním i vzhledem k ukončené softwarové podpoře rovněž zastarávají), a proto se jeví z pohledu vývojáře na rozdíl od iOS jako vhodnější alternativa.

9.1.3 Ubuntu Phone

Ačkoliv v prostředí základních a středních škol nenajde Ubuntu Phone nejspíše žádný prostor pro uplatnění, není možné jej ve výčtu opomenout. Jedná se v podstatě o jedinou zcela otevřenou alternativu k nejrozšířenějším platformám Android a iOS, která si prošla hektickým vývojovým obdobím stejně jako obdobím stagnace a pádu. Ovšem nyní se díky nadaci UBports[33] nachází opět ve stavu aktivního vývoje.

Na rozdíl od projektů Firefox OS či Windows Phone je Ubuntu Touch jeden z mála, který z pozdější řady podobných projektů zatím neskončil neúspěchem, a naopak zdá se, že komunita kolem něj je s rostoucím časem stále aktivnější. Podpora zařízení roste a zároveň dochází k četným softwarovým aktualizacím ve všech úrovních systému. (*UBports*)

Výhodou Ubuntu je jeho poměrná blízkost se systémem pro osobní počítače a tím i snazší užití celé řady aplikací a knihoven. Vývojáře může také zaujmout koncept konvergence, se kterým Canonical, firma stojící za vznikem projektu, přišel již v roce 2013, kdy Ubuntu pro telefony představil. Jedná se o schopnost využívat mobilní telefon po připojení k externímu monitoru jako běžný stolní počítač, přičemž konvergence v tomto významu neznamená jen uživatelské prostředí samotného Ubuntu, které se přizpůsobí, ale také podpora ze strany vývojářů aplikací, díky které je možné aplikace škálovat na různé obrazovky a přizpůsobovat pro ovládaní prsty i běžnou klávesnicí a myší.

Ubuntu je zde zmíněno hlavně jako ukázka alternativního systému, který může vývojáři poskytnout nejen dostatečně kvalitní nástroje pro vývoj svých aplikací, ale také poměrně velký zájem ze strany uživatelů, neboť na rozdíl od konkurence je ekosystém aplikací na Ubuntu Touch malý. To může vývojáře motivovat, neboť to v praxi znamená, že pokud přijde s jakoukoliv univerzálně použitelnou aplikací, získá si pro nedostatek podobných aplikací velice rychle své uživatele. (Kozub, *Nové Ubuntu z pohledu uživatele a vývojáře*, s. 58)

Samozřejmě u Ubuntu výčet nekončí. Existují i některé další a stále aktivní mobilní operační systémy, například Tizen[34] či Sailfish OS[35]. Některé jejich komponenty ale nesou, v kontrastu s Ubuntu Touch či čistým Androidem v podobě AOSP (bez nástrojů a doplňků Google)[36], navzdory otevřenému systému proprietární licence.

[33] *UBports Foundation*. Berlín: UBports Foundation, c2019. Dostupné také z: https://ubports.com/foundation/ubports-foundation.

[34] *Tizen* [online]. San Francisco (CA): Linux Foundation, c2012 [cit. 2019-04-04]. Dostupné z: https://www.tizen.org/.

[35] *Sailfish OS* [online]. Tampere (Finsko): Jolla, c2018 [cit. 2019-04-04]. Dostupné z: https://sailfishos.org/.

[36] *Android Open Source Project* [online]. Mountain View (California): Google, Open Handset Alliance [cit. 2019-04-04].

9.2 Příklady klasických vývojových prostředí

9.2.1 Qt Creator

Jedná se o vývojové prostředí, jehož první verze byla zveřejněna v roce 2009. Qt Creator je určený primárně pro tvorbu aplikací využívajících grafické rozhraní postavené na frameworku Qt. Samotné prostředí je možné spouštět na Linuxu, Windows i macOS.

Jeho nespornou výhodou je navzdory implementaci mnoha pokročilých funkcí jednoduché ovládání prostředí, podpora mnoha různých cílových platforem (desktop, Android, iOS[37] nebo i Ubuntu Touch), stejně jako skutečnosti, že si nástroj rozumí s různými jazyky a podporuje správu komplexních projektů. (*Supported Platforms*)

Vývojové prostředí je samo o sobě napsáno v C++, což může být také jeden z faktorů mající vliv na rychlý a bezproblémový chod prostředí v systému programátora. (*Qt Creator*)

9.2.2 Android studio

V roce 2013 Google představil a v roce 2014 zveřejnil své oficiální prostředí pro tvorbu aplikací pro systém Google Android. Nástroj je postavený na rozšířeném softwaru IntelliJ IDEA z dílem firmy JetBrains. To znamená, že je napsán v jazyce Java a je možné ho spustit jak ve Windows, tak na macOS i Linuxu. (*Android Studio*)

9.2.3 Xcode

Není možné vynechat ani oficiální nástroj pro tvorbu aplikací pro systémy Apple. Xcode tak umožňuje vývoj jak pro počítače se systémem macOS, tak pro mobilní zařízení s iOS a také chytré hodinky a televize Apple. Software je dostupný zdarma, ale zásadním problémem může být skutečnost, že je spustitelný pouze na systémech macOS.

9.3 Příklady prostředí podporujících blokovou editaci kódu programu

Pokud se učitel rozhodne učit žáky algoritmizaci cestou vývoje mobilních aplikací, přičemž bude jeho snahou odprosit žáky od samotného programovacího jazyka a soustředit se především na samotnou algoritmizaci, nemá k dispozici příliš velké možnosti. Aplikace, které toto poskytují, jsou prakticky pouze dvě a obě vznikly z jediného projektu.

Pro výběr mezi nimi je vhodné specifikovat základní kritéria, na jejichž základě je možné se i podle aktuálních potřeb rozhodnout:

[37] Podpora iOS je vázána na spouštění Qt Creatoru v systému macOS. V opačném případě není tato cílová platforma v nástroji k dispozici.

Tabulka 2: Kritéria hodnocení prostředí podporujících blokovou editaci kódu

Nástroj	Jazyk	Dostupnost IDE	Flexibilita	Podpora	Licence
MIT App Inventor 2	grafický	online	vázaný na Android	Komunitní fórum	otevřená
Thunkable	grafický	online	Android i iOS	Komunitní fórum	mix

9.3.1 MIT App Inventor 2

V roce 2010 zveřejnil Google cloudové řešení App Inventor pro tvorbu aplikací pro Android. Jak již z tohoto vyplývá, aplikace tak byla relativně nezávislá na prostředí uživatele, tedy na jeho operačním systému – stačil pouze moderní webový prohlížeč. Kromě toho usnadňovala vývoj pro mobilní zařízení v mnoha ohledech, neboť nevyžadovala instalaci a nastavování vývojového prostředí a především ani znalost konkrétního programovacího jazyka.

Základním rozhraním je plocha obrazovky, kam uživatel přesouvá jednotlivé komponenty z nástrojové lišty. Mezi ně patří například text, tlačítko nebo třeba i multimediální obsah, mapy či neviditelné komponenty jako je databáze. S nimi pak dále pracuje v blokovém editoru, kde je možné jednotlivým položkám přiřazovat funkce přetahováním jednotlivých bloků kódu ve vizuálním editoru podobně, jako v například jazyce Scratch. (Němec, *Naprogramujte si aplikaci pro mobilní telefon*)

Google se projektu ve druhé polovině roku 2011 vzdal a zveřejnil jeho zdrojové kódy. Aplikaci převzal pod svá křídla Institut MIT (Massachusetts Institute of Technology) a začal ji hostovat na svých serverech. V roce 2013 byl vydán App Inventor 2, který se uživatelsky změnil hlavně v blokovém editoru kódu, jež byl kompletně změněn z původního souběžně běžícího procesu v prostředí Java (Open Blocks) na systém běžící přímo v prohlížeči a využívající JavaScript (Blockly). (*App Inventor for Android*)

MIT App Inventor je dostupný pod licencí Apache-2.0 a MIT (*MIT App Inventor Public Open Source*), která je kompatibilní s licencí GPLv3.

9.3.2 Thunkable

Z MIT App Inventoru vznikl také projekt Thunkable, který v raných verzích vypadal a fungoval v podstatě shodně. Dnes je k dispozici kromě klasické verze také verze X, která rozšiřuje schopnosti původního editoru o export pro systémy iOS a také implementuje přepracované rozhraní, ačkoliv se z pohledu uživatelského rozhraní jedná především o facelift, neboť stále využívá například na Blockly postavený blokový editor kódu. Největší změnou prošla karta s designérem – došlo ke změně uspořádání komponent a celkový vzhled prostředí vypadá moderněji.

Podpora iOS v Thunkable X byla umožněna mj. změnou z generování aplikací postavených na Javě na aplikace využívající React Native, tedy JavaScriptový framework pro tvorbu nativních mobilních aplikací. Jedná se o mezivrstvu, která komunikuje s operačním systémem mobilního zařízení a navenek vytváří standardní uživatelské rozhraní, čímž se aplikace, která tento framework využívá, stává nerozeznatelnou od jiných nativních aplikací. (*Thunkable Cross Platform*)

Tabulka 3: Stručný přehled všech uvedených vývojových prostředí

Nástroj	**Licence**[38]	**Cílové platformy**	**Potřeba znalosti programovacího jazyka**
Qt Creator	LGPL	Multiplatformní	Ano
Android Studio	Apache 2.0	Android	Ano
Xcode	Freeware	systémy Apple	Ano
MIT App Inventor	Apache-2.0 a MIT	Android	Ne
Thunkable	Apache-2.0, MIT, closed-source	Android a iOS	Ne

10 Výběr vhodného nástroje

Protože z finančních i organizačních důvodů vychází užití mobilních zařízení lépe, než práce s dedikovanými zařízeními, rozhodl jsem se pro tuto kategorii. Rovněž výběr nástroje nebyl vzhledem k omezené nabídce obtížný. Ačkoliv mám sám zkušenosti s psaním kódu v mnoha jazycích, je ve školách bezesporu lepší začínat něčím snazším a to editory podporující blokové skládání jednotlivých částí programů jsou.

Při rozhodování se mezi MIT App Inventorem a Thunkable jsem se rozhodl využít nástroj MIT App Invertor i přes to, že má Thunkable větší podporu mobilních zařízení díky kompatibilitě s operačním systémem iOS. Tato výhoda je však pouze dočasná, neboť i MIT App Inventor plánuje v blízké budoucnosti zahrnout podporu pro mobilní zařízení s tímto systémem.

Mezi hlavní důvody patří skutečnost, že MIT App Inventor nevyvíjí firma, ale univerzita, jejíž zájmem není primárně svůj software nabízet komerčně, ale pomáhat rozšiřovat vzdělání. Budoucnost Thunkable není jistá, ovšem cílem autorů je na softwaru vydělávat. Ačkoliv to není zatím problém, může se stát, že budoucí verze Thunkable nebudou tak jednoduše dostupné jako ty dnešní. Navíc i když Thunkable staví na MIT App Inventoru, veškerá jejich rozšíření jsou uzavřená a nedostupná veřejnosti, což samo o sobě naznačuje, jakým směrem se chtějí vývojáři ubírat. (*MIT spin-out Thunkable hopes its drag-and-drop app builder can be a money-spinner too*)

Na druhou stranu jsou si oba nástroje stále ještě vizuálně natolik podobné, že je lze celkem bez potíží zaměňovat. Naneštěstí import původních projektů do Thunkable X, tedy nástupce předchozího Thunkable, jež s MIT App Inventorem sdílel nejen rozhraní editoru, ale i backend, není možný a stejně tak není možný ani jejich export. Tudíž neexistuje interoperabilita mezi oběma systémy a je v takovém případě potřeba aplikace vždy vytvářet znova.

10.1 Komunita kolem MIT App Inventoru

Samotný nástroj ale není jediným ukazatelem, rozhodujícím o vhodnosti či nezpůsobilosti užití programu ve výuce. Proto je velice důležité, že lze k MIT App Inventoru dohledat obrovské množství ukázek a návodů, které mohou učiteli pomoci, rozhodne-li se jej využít. (*App Inventor Tutorials*)

[38] Uvedené licence se týkají pouze samotného softwaru vyvíjeného jednotlivými firmami. Typicky kromě toho využívají další komponenty, zveřejněné pod otevřenými licencemi.

I přímo domovská stránka nástroje poskytuje jednoduché návody, pokrývající nejrůznější témata. K dispozici tak jsou ukázkové aplikace, včetně detailních tutoriálů doplněných o snímky obrazovek i samotné balíčky aplikací k jejich rychlému vyzkoušení. U tutoriálů pro úplné začátečníky je možné nalézt i názorná videa, v nichž je představen postup, stejně jako i funkční výstupy aplikací. (*Tutorials for App Inventor*)

Součástí projektu je také dokumentace[39], pokrývající témata jako jsou nastavení aplikace k tomu, abychom ji mohli používat, řešení problémů, popisy jednotlivých komponent, stejně jako univerzální přístupy a návrhy řešení některých obecnějších problémů.

V neposlední řadě odkazuje domovská stránka projektu na množství návodů určených přímo pro učitele. Ty většinou navrhují, jak aplikaci ve výuce využít a co do kurzů s ní zařadit. Je to místo pro sdílení těchto nápadů, ale i použitých materiálů. (*App Inventor for Educators*)

V rámci projektu existuje i komplexní fórum, kde je možné vyhledávat a sdílet potíže, podněty či vlastní zkušenosti s aplikací.

Kromě oficiálních on-line zdrojů je možné se o MIT App Inventoru dozvědět také z knih[40] či na některé z konferencí uvedených na webu programu. Rovněž je možné přihlásit se do některého z online webinářů. Alternativně je na internetu k dispozici celá řada dalších návodů, včetně video tutoriálů.

10.2 Prerekvizity práce s MIT App Inventorem

Podmínkou pro vstup do vývojového prostředí App Inventoru[41] je vyžadováno přihlášení se účtem Google. Pro samotné testování výsledného softwaru je pak ideální mít po ruce zařízení s operačním systémem Android, u nějž postačí, aby bylo samo připojeno na internet. V tomto případě je vyžadována instalace pomocného nástroje MIT AI2 Companion[42] přímo na zařízení s Androidem. Ten je dostupný v obchodě Google Play i jako samostatný instalační balíček na stránkách MIT App Inventoru.

Druhou možností je připojit zařízení kabelem USB, což již vyžaduje instalaci podpůrného softwaru aiStarter[43] přímo na pracovní počítač. aiStater se stará o propojení webového prohlížeče se zařízením Android a je dostupný pro operační systémy Windows, macOS a Linux.

Vytvořený program je nicméně bez fyzického zařízení možné otestovat také s pomocí emulátoru, nicméně pro tuto možnost jsou vyžadovány systémové změny, podobně jako v případě přímé komunikace prohlížeče a mobilního zařízení skrze USB kabel. Pro samotné žáky se jedná o nejméně vhodné řešení, neboť přicházejí o možnost bezprostředně si svou aplikaci vyzkoušet. Navíc relativní složitost ovládání grafického rozhraní Androidu v emulovaném prostředí a bez dotykové obrazovky

[39] *App Inventor Classic Reference Documentation* [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/content/reference-documentation.html.

[40] *Books* [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/books.html.

[41] *MIT App Inventor: IDE* [online]. Massachusetts: Massachusetts Institute of Technology, 2019 [cit. 2019-04-04]. Dostupné z: http://ai2.appinventor.mit.edu/.

[42] *MIT AI2 Companion* [online]. Mountain View (California): Google, c2019 [cit. 2019-04-04]. Dostupné z: https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3.

[43] *Installing and Running the Emulator in AI2* [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: https://appinventor.mit.edu/explore/ai2/setup-emulator.html.

si v případě absence fyzických zařízení žádá otázku, zda již v tomto případě není lepší použít jiný nástroj sloužící přímo k tvorbě aplikací určených pro desktopové počítače.

10.3 Prostředí MIT App Inventoru

Před zahájením práce s MIT App Inventorem je nezbytné seznámit se samotným prostředím. Učitel by měl mít před zahájením výuky dostatečné povědomí o tom, jak se s nástrojem manipuluje a jakými možnostmi prostředí disponuje.

Koncepce MIT App Inventoru je veskrze jednoduchá, a jak již bylo popsáno výše, skládá se ze dvou hlavních částí – tedy designér, který slouží primárně pro návrh vizuální stránky aplikace, ale i deklaraci neviditelných užitých komponent, a dále blokový editor kódu ne nepodobný Scratchi. (*Scratch*) Při založení nového projektu je výchozí obrazovkou právě designér.

Designový návrhář umožňuje vkládání jednotlivých komponent do zvolené obrazovky. Mezi komponenty se nacházejí prvky, jako tlačítko, popisek, seznam, formulářové prvky, kontejner pro zobrazení webového obsahu, mřížky pro rozložení obrazovky, multimediální přehrávací i nahrávací komponenty a překladač, kreslicí plocha Canvas, mapové podklady, nejrůznější senzory, objekty pro perzistentní ukládání dat do souborů, databází či cloudu i další nástroje pro sdílení obsahu. K dispozici je také rozšíření pro ovládání stavebnice LEGO® MINDSTORMS® a v neposlední řadě je možné prostředí doplnit o další rozšíření, která je možné stáhnout z webu.

Většina komponent, které uživatel do obrazovky přetažením z panelu vloží, obsahuje některá nastavení, které je možné upravit přímo v tomto prostředí, tj. bez nutnosti zásahu z editoru kódu. Například tlačítku či popisku je možné nastavit výchozí textovou hodnotu či vzhled. Logicky již ale není možné v tomto režimu nadefinovat, jaká událost se má spustit po kliknutí na tlačítko nebo v průběhu běhu programu zde nastavené hodnoty měnit bez jakékoliv programové logiky. K tomu je nezbytné přepnout se do blokového prostředí. Deklarace komponent v návrháři je nezbytná pro tvorbu smysluplné aplikace a je prerekvizitou pro jejich manipulaci v blokovém editoru kódu, neboť zde není možné pracovat s nedefinovanými prvky.

Tlačítka pro přesun mezi návrhovým a programovacím režimem se nacházejí v pravém horním rohu okna nástroje. Je důležité porozumět také oddělení obrazovek a tomu, že jak návrh, tak i programování vždy souvisí s konkrétní zvolenou obrazovkou. Kód vytvořený v rámci jedné obrazovky bude spuštěn jen a pouze tehdy, nachází-li se uživatel aplikace v daném pohledu. Zatímco při návrhu obrazovky je toto rozdělení typicky (z důvodu odlišné struktury aplikace) na první pohled patrné, u blokového editoru může docházet ke zmatečným situacím, neboť se zde člověk při větším množství obrazovek snadno splete. Přepínání mezi obrazovkami je realizováno naopak v levém horním prostoru okna aplikace, přičemž současně je zde k dispozici tlačítko pro přidávání obrazovek nových.

Podobně jako v režimu návrhu je i v blokovém editoru kódu k dispozici levý panel s nástroji, přičemž se zde nacházejí jednotlivé algoritmické konstrukce rozdělené do kategorií, stejně jako seznam komponent obrazovky odpovídající seznamu z návrhového režimu. Pro každý z objektů jsou k dispozici obslužné metody pro získávání i nastavování hodnot a vlastností těmto objektům, stejně jako bloky událostí, které realizují spouštění vnořených bloků po příslušné aktivitě.

Mezi algoritmickými konstrukty je možné nalézt prakticky cokoliv, co je potřeba – tj. k dispozici jsou podmínky, cykly, logické hodnoty, matematické funkce, nástroje pro práci s textem, globální i lokální proměnné, indexovatelné seznamy (pole) a procedury i funkce s argumenty i návratovou hodnotou. Rovněž jsou odděleně k dispozici barvy. V sekci Control, kde jsou seřazeny zmíněné podmínky a cykly, je možné nalézt také prvky specifické pro MIT App inventor, umožnující otevírání nových obrazovek a zavírání současné obrazovky nebo celé aplikace.

Ačkoliv může být skládání bloků v některých případech náročnější než prostý textový zápis, patří MIT App Inventor mezi prostředí se širokou základnou pro tvorbu aplikací, neboť poskytované nástroje nijak zásadně nelimitují fantazii vývojářů. Svědčí o tom i velké množství aplikací, které mezi sebou žáci sdílejí. Nad rámec toho jsou navíc každý měsíc posuzovány aplikace, které se přihlásí do soutěže, z nichž jsou pak vybírány aplikace od vývojářů třech věkových kategorií. (*App of the Month Winners*)

Část II

Akční výzkum

Východiskem praktické části práce je návrh aktivit pro rozvoj algoritmického myšlení, jejichž výstupem budou aplikace pro mobilní dotyková zařízení. K tomu, aby bylo možné určit, zda byly zvolené aktivity nastaveny správně a zda vedou žáky k rozvoji algoritmických dovedností, případně zda představují užitá zařízení či vybraný nástroj (MIT App Inventor) nějaké zásadní překážky, je nezbytné provést jejich ověření.

To ideálně vyžaduje splnění následujících podmínek:

1. Realizaci navržené výuky na zvolené střední škole.
2. Přítomnost hlavního vyučujícího a pozorovatele (např. druhý učitel), kteří budou oba pozorovat průběh hodin.
3. Aktivní zapojení žáků v procesu výzkumu - žáci budou na úlohách pracovat a bude od nich získána zpětná vazba.

Podle J. Hendla definované podmínky dobře pokrývá akční výzkum. Mezi jeho základní charakteristiky patří právě praktické ověření, sběr informací a závěrečná reflexe. Akční výzkum je typicky kvalitativního charakteru, což vyhovuje i navrženým představám ověření. J. Hendl dále uvádí potřebu stavět akční výzkum na předem získaných poznatcích, jež byly pro potřeby tohoto výzkumu shromážděny v teoretické části práce. Vzhledem k povaze šetření bude pozorování vedeno jako zúčastněné, což v tomto případě znamená, že se hlavní učitel bude aktivně ve výuce participovat a se žáky spolupracovat. (*Kvalitativní výzkum*, s. 138-139.)

J. Hendl poukazuje na to, že „zamýšlený výzkum je obvykle nutné před někým obhájit dříve, než se začne s jeho uskutečňováním. Například student potřebuje, aby plán práce schválil vedoucí práce nebo školitel“. Současně ale zmiňuje, že na rozdíl od konvenčního výzkum nemá kvalitativní výzkum žádný jasný předpis, na základě kterého by bylo možné podobný plán sestavit. Rovněž se předpokládá, že v průběhu výzkumu může docházet k jeho vývoji. Přesto takto předem definovaný nástin poslouží jako odrazový můstek. (*Kvalitativní výzkum*, s. 157.) I proto byl tento aspekt s vedoucí práce konzultován a průběh praktické části se od těchto konzultací bude dále odvíjet.

Jelikož se jedná o kvalitativní výzkum, je nezbytné uvažovat nad možnostmi sběru dat pro závěrečné zhodnocení. J. Hendl uvádí, že je pro takový výzkum důležité naslouchat vyprávění, klást otázky lidem a získávat od nich odpovědi. „Dotazování obecně zahrnuje různé typy rozhovorů, dotazníků, škál a testů.“ (*Kvalitativní výzkum*, s. 164.)

Výsledky výzkumu poslouží zejména jako podnět k budoucímu zlepšení testovaných aktivit. Jeho základní podstatou bude pozorování zvolené třídy v průběhu práce na aktivitách i během závěrečného setkání, kde žáci v rámci ověření předvedou své dovednosti v oblasti algoritmického myšlení prací na zadané testové úloze. Kromě hlavního vyučujícího, který povede výuku a bude simultánně pozorovat její průběh a reagovat na případné potíže, poslouží ke sběru dat i druhý vyučující v roli pozorovatele. Nedílnou součástí budou i reakce samotných žáků v průběhu hodin i bezprostředně po dokončení práce a souhrnně po realizaci všech aktivit. K tomu budou vytvořeny jednoduché

dotazníky, které jim umožní aktivity snadno zhodnotit a současně poskytnou prostor pro případné doplňující podněty.

Pro návrh úloh a následnou výuku algoritmického myšlení v prostředí MIT App Inventoru je rovněž nezbytné znát cílovou skupinu žáků. Obtížnost úloh je pak vhodné nastavit zejména na základě aktuálních znalostí cílové skupiny. Je důležité vědět, zda jde o žáky mírně nebo středně pokročilé či snad úplné začátečníky. Níže definované aktivity tak počítají se žáky, kteří v prostředí MIT App Inventoru nikdy nepracovali. Hodiny je rovněž potřeba uzpůsobit jejich znalostem v oblasti algoritmizace a u pro žáky nových témat vyčlenit prostor k jejich vysvětlení.

Předpokládá se, že na základě připravených úloh dokáží žáci po jejich realizaci ovládat probírané učivo a pracovat tak na tvorbě aplikací samostatně. Pro každou z těchto úloh jsou známy jejich konkrétní předpoklady, jež jsou vystiženy uvedenými cíli. Shrnuty jsou i předpokládané potíže, pro které jsou navržena možná řešení.

11 Předpokládaný průběh akčního výzkumu

Akční výzkum bude rozdělen do následujících fází:

- Výběr témat pro práci žáků s MIT App Inventorem tak, aby zvolená témata pokryla všechny základní koncepty algoritmizace, zejména podmínky, funkce, proměnné, cykly.
- Návrh jednotlivých úloh a jejich časového rozvržení s ohledem na současně definované cíle.
- Příprava podkladových materiálů na základě samostatně ověřených úloh.
  - Námět na realizaci s některými otevřenými možnostmi.
  - Podrobný návod pro realizaci úloh popisující jednotlivé kroky řešení.
  - Elektronická podpora v prostředí Moodle, jakožto prostor, kde žáci naleznou zadání i řešení jednotlivých úloh. Současně Moodle kurz poslouží pro odevzdání závěrečné ověřovací aktivity.
- Výuka na zvolené škole, rozdělená do potřebného časového období. V rámci ní dojde k nezávislému pozorování druhým učitelem, pozorování z pohledu vedoucího vyučujícího a ke sběru dat přímo od žáků na základě anonymního dotazníku.
- Realizace zakončující testové úlohy, jejíž cílem je ověřit jak moc či zda vůbec předchozí aktivity vedly k rozvoji schopností práce s nástrojem a související algoritmizací.
- V závěru dojde k vyhodnocení získaných dat a k jejich interpretaci včetně uvedení doporučení, která z případných nedostatků plynou.

12 Přehled zvolených aktivit

Tabulka 4: Přehled všech zvolených aktivit k realizaci

Pořadí	**Název aktivity**	**Celkový čas**	**Náročnost** [44]
1	Tvorba úvodní aplikace	1 vyučovací hodina	★
2	Pohyb kuličky po obrazovce	2 vyučovací hodiny	★★
3	Hrátky s různými typy proměnných, polem a grafickým seznamem	65 – 70 minut (necelé 2 vyučovací hodiny)	★★★
4	Záznam zvuku s možností přehrávání	100 – 110 minut (dvě nebo necelé 3 vyučovací hodiny)	★★
5	Hra s raketou	4 vyučovací hodiny	★★★
6	Závěrečná ověřovací úloha	1 vyučovací hodina	★★

[44] ★ = snadné, ★★ = středně náročné, ★★★ = náročné

13 Požadavky společné pro všechny úlohy

13.1 Technické vybavení

- Učitel – Potřebuje počítač s internetovým připojením. Dále ideálně dataprojektor či jiný způsob prezentace pro živé promítání vlastní práce a ukázek s MIT App Inventorem. Užitečná, avšak nikoliv nezbytná, je rovněž interaktivní tabule pro možné větší zapojení žáků při hledání společného řešení problémů. Učitel musí mít k dispozici také mobilní zařízení nebo emulátor s internetovým připojením pro spuštění výsledných aplikací přímo v rámci výuky, ale i k předchozí přípravě.
- Žák – Podrobně jako učitel musí mít během hodin, i pro případnou práci na domácích úkolech, k dispozici osobní počítač s internetovým připojením. Současně také mobilní zařízení s internetovým připojením pro spuštění a testování výsledných aplikací.

13.2 Softwarové vybavení

- Webový prohlížeč Mozilla Firefox, Apple Safari či Google Chrome. Microsoft Internet Explorer ani Microsoft Edge nejsou oficiálně podporovány, ačkoliv druhý zástupce není explicitně zmíněn a vzhledem k podpoře novějších standardů je pravděpodobné, že fungovat bude. Zvláště pak v nadcházejících verzích využívající stejné jádro jako samotný Google Chrome. V době psaní diplomové práce uváděl oficiální zdroj jako minimální verze prohlížečů tato, z dnešního pohledu již velmi stará, vydání:
  - Mozilla Firefox 3.6 nebo novější
  - Apple Safari 5.0 nebo novější
  - Google Chrome 4.0 nebo novější [45] [46]

  Z praktického ověření nicméně vyplynulo, že i nejnovější verze prohlížeče Google Chrome mohou mít při spouštění aplikací na zařízení Android potíže. Ve všech případech se jednalo o chybu s nic neříkající hláškou „An internal error has occurred“. Při použití prohlížeče Firefox se nicméně podařilo aplikaci na mobilním zařízení spustit.
- Operační systém Android nainstalovaný na zařízení či v emulátoru. V době psaní diplomové práce byla jako minimální verze OS Android stanovena verze 2.3 „Gingerbread“ [2]
- Aplikace MIT AI2 Companion nainstalovaná na mobilním zařízení. Tu je možno stáhnout přímo z Google Play nebo na stránkách nástroje. Aplikace je tak dostupná i pro zařízení Android bez služeb Google Play.[47]
- Oficiální stránky projektu dále zmiňují závislosti na operačním systému a připouštějí vybrané verze macOS, Windows a Linuxu. Vzhledem k tomu, že v základním užití, tj. při kombinaci webový prohlížeč a mobilní zařízení, při použití kódu a internetové sítě pro nahrávání aplikace do zařízení není potřeba instalace žádného dodatečného softwaru a vše běží v cloudu, je omezení na operační systém nepodmíněné. Platí tak, že uživatelé jiných operačních systémů

[45] Požadavky pro editor kódu Blockly: *Supported browsers and versions for Blockly web?* [online]. Mountain View (California): Google, 2017 [cit. 2019-04-04]. Dostupné z: https://groups.google.com/d/msg/blockly/o3pERaRQhSg/eVyheZtvAQAJ

[46] Požadavky pro MIT APP Inventor: *System Requirements* [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://explore.appinventor.mit.edu/content/system-requirements

[47] *Connect your Phone or Tablet over WiFi* [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/ai2/setup-device-wifi.html.

mohou službu využívat úplně stejně, pokud jejich systém podporuje některý z podporovaných webových prohlížečů, případně také neoficiálně, je-li jejich prohlížeč dostatečně kompatibilní pro běh nástroje. Na základě vlastních testů tak není problém spustit nástroj například také přímo na mobilním zařízení.

Minimální softwarové požadavky udávané autory programu nejsou vždy zcela přesné a vždy je nejlepší si před samotnou prací otestovat, zda daná konfigurace funguje, či ne. MIT App Inventor je navíc pravidelně aktualizován a postupně jednotlivé chyby opravuje – což znamená, že pokud na nejnovějším zařízení aplikaci nebylo doposud možné spustit, může být po aktualizaci některé ze součástí MIT App Inventoru vše jinak.

13.3 Účty cloudových služeb

- Google účet pro přístup do cloudového nástroje MIT App Inventor. Bez tohoto účtu není možné vůbec aplikace začít vyvíjet.

13.4 Vyučovací metody

V průběhu výuky bude využita kombinace více metod s cílem zaměřit se na co nejvíce smyslů žáků. Ideálním průběhem hodin pak bude takový průběh, který bude pro žáky zážitný a v němž budou moct rovněž uplatnit i své vlastní nápady.

Mezi klasickými metodami se pak jedná především o metodu slovní, která je nezbytná pro předávání základních informací, na základě kterých teprve mohou žáci přistupovat k dalším krokům. Zde spadá především vysvětlování. Dále navazuje metoda dialogická, která aktivně zapojuje žáky do problému. Uplatní se zde především metoda rozhovoru, například při ověřování úrovně znalostí či ke zjištění, zda žáci pochopili určitou situaci. Velkou roli hrají také metody názorně demonstrační, při nichž učitel může žákům představit nejrůznější funkcionality, objasnit jak které prvky pracují nebo jim pouze ukázat, jak má vypadat výsledné řešení, čeho se má dosáhnout tak, aby v tomto směru nemohly vzniknout pochyby jinak obecně možné při nedostatečném slovním vysvětlení.

V rámci členění podle pohledu způsobu poznávání jsou kromě metod slovních a názorně demonstračních zařazeny ještě metody praktické, které budou hlavní náplní výuky. Jedná se o samotnou práci žáků na tvorbě vlastních programů a jejich následné testování. (Skalková, *Obecná didaktika*, s. 181-185)

Opomenuta by neměla být ani diskuze, jež je součástí metod aktivizujících. Zejména pak v kontextu rozpravy nad problémem a jeho řešením. Diskuze by měla vést žáky k přemýšlení o situaci a umožnit jim proniknout hlouběji do problému. Mohou v ní přinášet vlastní nápady, které spadají rovněž do brainstormingu uvedeného výše, ale především se pak dělit i o vlastní poznatky či popsat místa, v nichž měli potíže. To otevře prostor pro sdílení vědomostí mezi žáky a učitelem a pomůže žákům srovnat se na bližší úroveň.

Protože je náplní práce rovněž i vlastní bádání, na jehož počátku nemají žáci všechny informace k řešení problému, spadá mezi užité vyučovací metody rovněž i metoda problémová. Stejně tak je zastoupena při hledání lepších řešení, tedy součástí optimalizace.

Z komplexních metod pak stojí za zmínku výuka podporovaná počítačem, při níž žáci nejen že mohou hledat řešení na internetu a v rámci nápovědy MIT App Inventoru, ale veškerou svou vytvořenou práci si mohou v podstatě okamžitě ověřit tak, že ji nahrají do mobilního zařízení, kde vlastní aplikaci uvidí, mohou s ní interagovat a zjišťovat, jak se za různých okolností jimi vytvořený algoritmus či algoritmy chovají. Užitečným je také brainstorming, jehož cílem je přimět žáky navrhnout vlastní postup pro řešení určitého problému.

Nezanedbatelnou roli hraje při výuce algoritmizace rovněž samostatná práce žáků, která je vhodným prostředkem pro zapamatování si poznatků. (Maňák et al., *Výukové metody*, s. 53-187)

13.5 Pedagogické zásady a principy

- Zásada názornosti – Učitel sám názorně představuje prvky potřebné k sestavení aplikace, případě společně se žáky celou aplikaci během výuky, ovšem s aktivním zapojením žáků do hodiny, tvoří.
- Zásada emocionálnosti – Žáci vytvářejí aplikace v prostředí, kde mohou přinášet vlastní nápady a ty následně realizovat. Učitel žákům pomáhá s případnými komplikacemi a radí jak správně postupovat. Cílem tak je zanechat v žácích především pozitivní zážitky.
- Zásada individuálního přístupu – Ne každý žák musí nutně splnit celou úlohu, tzn. že je potřeba respektovat potřeby a tempo jednotlivých žáků. Smyslem výuky je seznámit všechny s užitými algoritmickými konstrukty a naučit je tvorby základních mobilních aplikací v prostředí MIT App Inventoru.
- Zásada přiměřenosti – Podstata všech aplikací je sama o sobě veskrze jednoduchá a odpovídá věku žáků středních škol. Učitel by měl přizpůsobit náročnost v závislosti na tom, zda je výuka prováděna v nižších či naopak vyšších ročnících. Stejně tak je třeba respektovat individuální schopnosti žáků i celé třídy, obdobně jako současnou atmosféru třídy.
- Zásada aktivnosti – Žáci jsou co nejvíce vtaženi do hodiny, zapojováni za účelem motivace. Tvorba aplikací je diskutována a průběh ideálně upraven podle momentálních nápadů třídy tak, aby si žáci sami přicházeli na co největší množství řešení sami a ty dále mohli aplikovat.
- Zásada soustavnosti – Učitel vede žáky podle nastaveného plánu a snaží se systematicky postupovat v jeho plnění. Učivo na sebe musí logicky navazovat.
- Princip zpětné vazby – Učitel získává zpětnou vazbu jakožto podnět pro úpravu výuky na základě aktuálních nedostatků či naopak znalosti žáků o daném tématu.
- Princip spojení školy se životem – Výuka je koncipována tak, aby zahrnovala aktuální požadavky společnosti, tedy rozvoj algoritmizace, ale současně i obecně digitální gramotnosti žáků. (Vlčková, *Pedagogické zásady*; Polák, *Pedagogické zásady a principy*)

14 Navržené aktivity

14.1 Úvodní aplikace (1. aktivita)

Prvním krokem nezbytným pro práci s MIT App Inventorem je potřeba seznámit žáky se základy prostředí aplikace, zvláště pak s tím, jaké jsou k dispozici nástroje a co je potřeba k tomu, aby vstupní aplikace fungovala jak z grafické, tak především logické strany.

K tomuto účelu může posloužit jednoduchá aplikace, která může spočívat například pouze v tlačítku umístěném na obrazovce, přičemž po kliknutí na něj se ozve zvuk, změní barva pozadí aplikace či nastane nějaká jiná dobře pozorovatelná akce.

Ať již si učitel zvolí libovolný postup, ideálně ušitý na míru konkrétní skupině žáků, je v tomto kroku nezbytné dodržovat potřebu vyhnout se náročnějším konstruktům a skutečně se držet pouze jednotlivosti. O to více jsou-li cílovou skupinou žáci obecného vzdělávání, bez znalostí programování.

14.1.1 Vstupní požadavky na žáka

- Žáci znají pojem algoritmus a umějí ho použít.
- Žáci jsou na počítači schopni spustit webovou aplikaci MIT App Inventor.
- Žáci umí na mobilním zařízení spustit aplikaci MIT AI2 Companion a v ní dále vytvořený program.

14.1.2 Obecný cíl

Žák samostatně, po nezbytném úvodu a představení prostředí MIT App Inventoru, sestaví strohou kostru aplikace s užitím prvku tlačítko doplněnou o jednoduchou logiku vedoucí k viditelné změně a dokáže ji prezentovat na obilním dotykovém zařízení.

14.1.3 Konkrétní cíle

- Žák použije nové grafické prvky (zejména pak tlačítko) prozkoumá a ručně upraví jejich vlastnosti.
- Žák sestaví logické bloky v blokovém editoru kódu tak, aby obsloužil událost tlačítka.
- Žák použije volání obslužných metod grafických prvků v programovém prostředí jako alternativní cestu pro úpravu jejich vlastnosti.

14.1.4 Postup

Tabulka 5: Průběh úvodní aktivity

Čas	**Popis aktivity**	**Vyučovací metody**	**Užité prvky**
5 - 10 minut	Učitel uvede žáky do tématu, stručně je seznámí s MIT App Inventorem (jeho součásti a požadavky pro práci s ním) a předvede již hotovou jednoduchou aplikaci na svém mobilním zařízení.	Slovní, (dialogická metoda, názorně demonstrační metoda)	–
5 minut	Učitel promítne QR kód s klíčem k demo aplikaci na projektoru a vyzve žáky k tomu, aby si každý spustili aplikaci MIT AI2 Companion na svých zařízeních a aplikaci si spustili a vyzkoušeli. Tím se ověří, že jsou žáci připraveni testovat i své vlastní aplikace.	Slovní	–
25 minut	Žáci jsou následně vyzváni k tomu, aby se sami přihlásili do MIT App Inventoru a začali prací na novém projektu. Společně s učitelem budou postupně objevovat jednotlivé části prostředí, především pak obrazovky návrhu a kódu a k nim příslušné nástrojové lišty. Během toho vloží do prostředí tlačítko, pojmenují ho, změní mu text a případně další vlastnosti a v blokovém editoru kódu mu přiřadí akci – pro začátek například změnu pozadí hlavní obrazovky. Finální aplikaci nahrají do svých zařízení.	Dialogická metoda, (instruktáž, brainstorming)	Screen, Button
5 minut	Závěr hodiny je vhodné vyhradit diskusi nad tím, co by bylo možné s aplikací dále dělat a jak ji rozšířit. Cílem této části je motivovat žáky k další práci s prostředím.	Diskuze	–
Doporučení v případě rychlejšího průběhu	Učitel upraví zadání tak, aby aplikace po kliknutí na tlačítko zahrála například melodii (tj. dojde k vložení nového neviditelného prvku **Player** či **Sound** do obrazovky aplikace, nahrání audio souboru do projektu z počítače a následné navázání akce na tlačítko). Popřípadě je možné měnit barvu pozadí náhodně či cyklicky přepínat mezi výchozí barvou obrazovky a alternativní. Finální aplikaci nahrají do svých zařízení.	Slovní metoda, (instruktáž, badatelská)	Button, dále např. Player / Sound nebo Screen

14.1.5 Výstupy z aktivity

Žáci by měli být po uplynutí vyučovací hodiny schopni spustit MIT App Inventor, přihlásit se do něj a v něm vytvořené aplikace nahrávat do mobilního zařízení. Rovněž by měli mít schopnost vkládat do prostředí hlavní obrazovky nové prvky a pracovat s nimi v blokovém editoru kódu.

Nezbytná je schopnost spouštět sekvence kódu v návaznosti na události – například po kliknutí na tlačítko. Osvojení pokročilejších prvků, z počátku pak zejména podmínek, je závislé na průběhu hodiny, zvolené ukázkové aplikaci a především pak časových možnostech.

14.1.6 Námět na domácí úkol

V této části není nezbytné zadávat žákům domácí úkol, ale je možné jej zařadit a motivovat žáky aktivitou. Jako možnost se nabízí vyzvat žáky například k tomu, aby přidali do aplikace textové pole pro uživatelský vstup a po kliknutí na tlačítko aplikace tento vstup nahlas přečetla s pomocí prvku **TextToSpeech**.

14.1.7 Možné potíže

- Žáci se nepřipraví na hodinu a nebudou mít na svém zařízení nainstalován nástroj MIT AI2 Companion ke spouštění vytvořených aplikací.
  - Učitel žákům zapůjčí školní zařízení, je-li to možné, případně nechá žáky pracovat po skupinách.
- Žáci nevlastní mobilní zařízení s OS Android.
  - Učitel buďto použije školních zařízení, jsou-li vhodná k dispozici, nebo vytvoří či nechá žáky vytvořit skupinky tak, aby v každé bylo alespoň jedno kompatibilní zařízení. Mohou pak pracovat společně na jednom projektu, nebo každý samostatně, pouze si zařízení k testování aplikace musejí nechat kolovat.
- Dojde k chybám při nahrávání aplikací z počítačů na mobilní zařízení.
  - Je potřeba zajistit kvalitní internetové připojení pro všechny žáky ve třídě, popřípadě nainstalovat App Inventor Setup, který umožňuje propojit zařízení s použitím kabelu a nahrávat software skrze něj.[48]
    Při tomto použití je nezbytné, aby bylo rovněž k dispozici dostatek kabelů pro připojení zařízení.
- Pokud bude učitel zadávat samostatné úkoly, je vhodnější, aby každý žák pracoval na svém vlastním projektu. Žáci, kteří nemají zařízení s OS Android, musí mít možnost pracovat se školním zařízením a tudíž jim musí být i pro případné úkoly zapůjčen vhodný přístroj.
  Pokud nemají všichni žáci přístup k OS Android a není v kapacitách školy přístroje zapůjčit i na doma, není vhodné domácí úkoly zadávat. Na druhou stranu při práci ve skupinkách je i v takovém případě možné, aby žáci své projekty zdokonalovali společně mimo výuku, například v rámci volných hodin přístupu k počítačům, ovšem tento způsob práce nemůže být logicky podmínkou.

[48] *Connecting to a phone or tablet with a USB cable* [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/ai2/setup-device-usb.html.

- MIT App Inventor při použití komponenty Sound nepracuje zcela intuitivně a audio soubor, který je jí předložen skrze návrhář prostředí, aplikace nedokáže na všech zařízeních přehrát. Řešením je buď využít komponentu Player, nebo komponentě Sound (například při inicializaci obrazovky) tento soubor přímo nadefinovat v blokovém editoru kódu.

14.1.8 Námět na realizaci

V ukázce níže je pro seznámení se s prostředím MIT App Inventoru využito pouze prvku tlačítka, přičemž na základě uživatelského podnětu dochází ke změně barvy pozadí plochy aplikace.

Na hlavní obrazovce tak bude vloženo a vycentrováno jedno tlačítko s popiskem například „Klikni na mě“, pojmenované třeba „baseButton“.

V tomto případě není nezbytné dodržovat anglické názvy objektů a proměnných. Mezi programátory se však jedná o rozšířenou praxi a z pohledu žáků střední školy není užití angličtiny problém. Podobně není potřeba dodržovat psaní ve formátu camelCase, kde až na první každé další slovo začíná velkým písmenem a nevyužívá se mezer.

Učitel si tedy může zvolit, jaký způsob využije – měl by být ale v průběhu všech lekcí konzistentní. Současně by měl dát žákům na výběr, ať si zvolí vlastní přístup, a vysvětlit jim možná úskalí práce s diakritikou a mezerami. Přímo MIT App Inventor nicméně dokáže s diakritikou pracovat, ovšem mezery okamžitě převádí na podtržítka.

Soustředí-li se aplikace pouze na změnu barvy pozadí, jedná se o triviální úlohu, na které je ale možné ukázat základní práci s blokovým editorem kódu. Učitel se tedy společně se žáky přepne do módu editace programu a sestaví následující sekvenci. Pro přehlednost je volen pseudokód:

Algoritmus 1 Příklad řešení změny barvy pozadí aplikace po kliknutí na tlačítko

```
when baseButton .Click do {
        set Screen1.BackgroundColor to "zelená" // např. zelená
            barva ručně zvolená v paletě barev
}
```

Kód nyní využívá události akce a přiřazení hodnoty do vlastnosti objektu obrazovky.

Název hlavní obrazovky „Screen1“ není ve výše uvedené ukázce změněn na vhodnější jako třeba „homeScreen“ nebo „mainScreen“. Důvodem je skutečnost, že MIT App Inventor sám o sobě neumožňuje změnit název již vytvořené obrazovky a uživatel se tak musí spokojit v tomto případě s předdefinovaným názvem.

Nyní vzniká prostor k tomu, aby si žáci mohli takto vytvořené aplikace nahrát do zařízení s předinstalovanou aplikací MIT AI2 Companion. To buď s pomocí kamery a QR kódu, nebo manuálně vložením kódu z obrazovky.

Aby bylo možné aplikaci do zařízení zaslat, je třeba využít tlačítko menu „Connect“ v horní části obrazovky MIT App Inventoru a zvolit položku „AI Companion“.

Po spuštění aplikace na mobilních zařízeních bude zobrazena standardní bílá obrazovka s tlačítkem „Klikni na mě“. Když na něj žáci kliknou, změní se barva pozadí obrazovky na zelenou nebo jinou vybranou barvu.

Dále je možné aplikaci rozšířit o změnu barvy nazpět. Ukládání stavu zatím neprobíhá s užitím proměnných, takže je třeba tázat se objektu obrazovky „Screen1“ na aktuální barvu pozadí aplikace. Pokud bude již např. zelená, provede kód aplikace změnu na bílou. V opačném případě přepne na zvolenou jinou. Díky tomu se bude pozadí aplikace cyklicky měnit po každém klepnutí na tlačítko.

Algoritmus 2 Možná cyklická změna barvy pozadí aplikace po opětovném kliknutí na tlačítko

```
when baseButton .Click do {
        if(Screen1.BackgroundColor == "bílá") {
                set Screen1.BackgroundColor to "zelená"
        }
        else {
                set Screen1.BackgroundColor to "bílá"
        }
}
```

V tomto kroku již bylo nezbytné zařadit konstrukt podmínky i porovnání.

Protože ale po startu aplikace není barva pozadí bílá, nýbrž je její hodnota odlišná (je-li nastavena bílá, je ve skutečnosti barva pozadí obrazovky výchozí, tj. nedefinovaná hodnota), je třeba ještě po inicializaci okna nastavit tuto barvu v kódu tak, aby porovnání fungovalo i po prvním kliku na tlačítko.

Algoritmus 3 Ukázka definice výchozí barvy pozadí v prostředí kódu

```
when Screen1 .Initialize do {
        set Screen1.BackgroundColor to "bílá"
}
```

Možným dalším rozšířením je úprava aplikace tak, aby po kliknutí na tlačítko přehrála krátkou melodii. K tomu lze využít komponentu **Sound** v návrhu aplikace v sekci „Media“, která je na rozdíl od objektu **Player** navržena pro kratší zvuky. Na druhou stranu u komponenty **Sound** je při běhu na některých zařízeních nezbytné nastavit cestu k souboru z programového prostředí, neboť čistě při nastavení v designéru není komponenta správně inicializovaná.

Učitel nechá žáky, ať si z internetu stáhnou některou z melodií či zvuků. Upozorní je na to, že se z ní přehraje jen krátký úsek zvolený v milisekundách (např. 5000 ms). Při hledání vhodných nahrávek je důležité žáky upozornit na to, že by neměli stahovat hudbu, která není volně šiřitelná, a pro tento účel je odkázat například na server[49], jež obsahuje širokou nabídku zvukových efektů a nahrávek dostupných pod různými svobodnými licencemi. Alternativně jsou-li k dispozici nějaká média již na lokálním počítači, je možné ušetřit si hledání a využít rovnou ta.

Do obrazovky v designéru je dále nezbytné přetáhnout neviditelnou položku **Sound**, pojmenovat ji např. „baseButtonSound“ a nastavit interval podle vlastního uvážení, ale ideálně ne příliš dlouhý. V poli `Source` je dále třeba zvolit a nahrát příslušný audio soubor z počítače.

[49] *Freesound* [online]. Barcelona: Universitat Pompeu Fabra, 2019 [cit. 2019-04-04]. Dostupné z: https://freesound.org/.

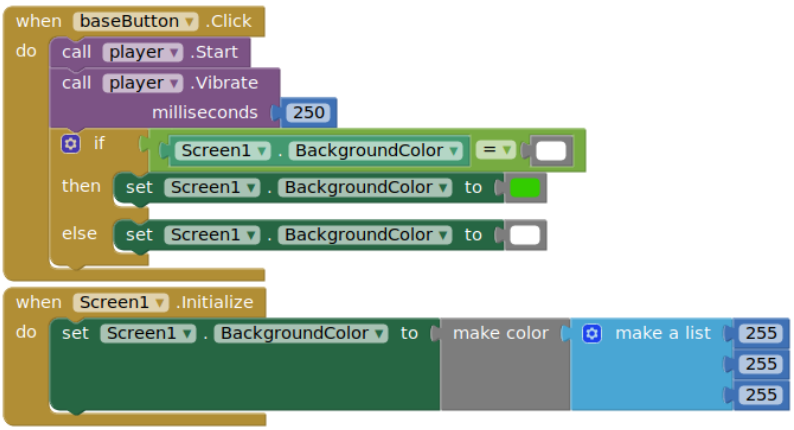
Zde je rovněž třeba dávat pozor na podporované formáty[50].

V blokovém editoru kódu je pak potřeba již jen zavolat příslušnou metodu `.Play` po kliknutí na tlačítko. Pro větší efekt lze rovněž nechat zařízení po zvolenou dobu vibrovat. K tomu je nezbytné, aby byl vibrační mini motor v zařízení obsažen. V opačném případě volání této metody nebude mít žádný účinek.

Algoritmus 4 Příklad přehrání hudby i spuštění vibrací po kliknutí na tlačítko

```
when baseButton .Click do {
        call baseButtonSound .Play
        call baseButtonSound .Vibrate millisecs 500 // Vloženo a
            upraveno číslo "0" ze sekce Math
        if(Screen1.BackgroundColor == "bílá") {
                set Screen1.BackgroundColor to "zelená"
        }
        else {
                set Screen1.BackgroundColor to "bílá"
        }
}
```

Takto vytvořený projekt je možné uložit buď jako nový, nebo si pouze vytvořit checkpoint (verzi), na které se bude dále stavět.



Obrázek 2: Příklad jednoduchého programu měnícího barvu pozadí aplikace s ukázkou sestavení barev pomocí pole

[50] *Supported media formats* [online]. Mountain View (California): Google, 2017 [cit. 2019-04-04]. Dostupné z: https://developer.android.com/guide/topics/media/media-formats.

14.2 Pohyb kuličky po obrazovce (2. aktivita)

Podstatou aktivity je tvorba jednoduché hry, v níž má hráč k dispozici kuličku a například náklonem obrazovky se ji snaží dostat k cílové hranici, přičemž se zároveň musí vyhýbat překážce.

Základem hry může být již existující aplikace, vzniklá v první lekci, popřípadě je možné začít zcela od počátku. Výchozím stavem by však měla být primární obrazovka s tlačítkem, které následně po kliknutí na něj přenese uživatele na novou stránku obsahující samotnou hru.

14.2.1 Vstupní požadavky na žáka

- Žáci mají znalosti potřebné pro předchozí aktivitu.
- Žáci dokáží pracovat s MIT App Inventorem na takové úrovni, že dokáží vkládat nové prvky do obrazovky aplikace, upravovat je a využívat funkcí blokového editoru kódu ke skládání logických struktur.
- Žáci jsou schopni vytvořenou aplikaci spustit na mobilním zařízení.

14.2.2 Obecný cíl

Žák s vedením učitele na základě hodnot senzoru orientace docílí naklonění odpovídajícímu pohybu grafického prvku zapouzdřeného v **Canvasu** po obrazovce a dále samostatně, po představení nezbytných prvků v prostředí MIT App Inventoru, detekuje kolizi dvou objektů za účelem změny skóre.

14.2.3 Konkrétní cíle

- Žák sestaví responzivní aplikaci, tj. nezávislou na rozlišení i poměru stran zařízení.
- Žák použije událost senzoru orientace a data o náklonu využije ke změně směru předmětu na obrazovce. Alternativně, není-li senzor v zařízení k dispozici, úplně stejným způsobem využije metodu `.Flung` pro odečítání informací z dotyků prstů.
- Žák detekuje objekt, do kterého pohybující se předmět narazil.
- Žák na základě analýzy kódu a testování aplikace obhájí správnost řešení, opravuje chyby, diskutuje nad možnými alternativami a zvažuje vylepšení.

14.2.4 Vyučovací zásady

- Zásada trvalosti – Žáci se vracejí k již probranému a opětovným užitím, případně mírným rozšiřováním poznatků, si dovednosti prohlubují a upevňují. Učitel opakuje kostru probírané látky z minulých hodin a dbá na její procvičování.

14.2.5 Postup

Tabulka 6: Průběh aktivity s kuličkou

Čas	**Popis aktivity**	**Vyučovací metody**	**Užité prvky**
5 minut	V úvodu učitel představí problém, který budou žáci řešit, a vysvětlí jim, co je jejich cílem.	Slovní metoda	–
5 – 10 minut	Na základě znalostí žáků z předchozí hodiny i skutečnosti, že budou mít před sebou k dispozici webový nástroj MIT App Inventor, dojde k diskuzi žáků a učitele nad tím, jakým způsobem by bylo možné dojít k cíli, jak úlohu řešit. Tato část volně navazuje na první a jejím výstupem může být například kresba námětu aplikace (soupis potřebných prvků a jejich vazeb) na tabuli.	Diskuze	–
20 – 25 minut	Žáci společně s učitelem vytvoří na hlavní obrazovce tlačítko, pracují-li od začátku, popřípadě rovnou přeskočí ke kroku vložení nové obrazovky do aplikace. Dále pak tlačítku na hlavní obrazovce nastaví akci přesunutí na novou obrazovku. Prvky v nové obrazovce rozvrhnou tak, aby bylo možné vrátit se dalším tlačítkem zpět na úvodní stránku i začít jiným tlačítkem novou hru. Na plochu rovněž vloží **Canvas** a do něj 3x **Ball** – jeden jako pohyblivý, další jako cíl a poslední jako past a nastaví jim různé velikostní i barevné vlastnosti. Dále jako neviditelný prvek vloží, podporuje-li jejich zařízení rozpoznání orientace, příslušný senzor pro čtení těchto dat.	Názorně demonstrační metody, (instruktáž, dialogická metoda)	Screen, Button, Canvas + Ball, volitelně rovněž Orientation-Sensor
20 - 25 minut	V blokovém editoru vloží funkci pro novou hru, jejíž cílem bude rozmístit prvky náhodně. K tomu bude rovněž užitečná sada dvou funkcí pro získání náhodného čísla pro X a Y. Při změně orientace zařízení pak nastaví odlišnou orientaci pohybu, ale i rychlost v závislosti na vychýlení, příslušnému pohyblivému prvku **Ball**.	Instruktáž, (metoda zkušenostního učení, dialogická metoda, názorně demonstrační metody)	Orientation-Sensor, Ball

20 minut	Žáci vytvoří algoritmus, který po kolizi pohyblivého objektu **Ball** s příslušnými statickými prvky **Ball** způsobí vyvolání nové hry. Rovněž v tomto okamžiku dojde k navýšení či snížení skóre.	Instruktáž, (metoda zkušenostního učení, dialogická metoda, názorně demonstrační metody)	Ball
2 minuty	Závěr hodiny je vhodné vyhradit rychlou reflexí a diskusi nad tím, co by bylo možné déle rozšířit.	Diskuze	–

V případě, že není v zařízení senzor orientace přítomen, je možné využít volání `.Flung`, které umožňuje posunout balón jeho odpálením prstem po obrazovce. Rovněž je možné zkombinovat oba přístupy a `.Flung` využit jen a pouze tehdy, není-li senzor orientace přítomen.

14.2.6 Výstupy z aktivity

Po dokončení lekce by žáci měli být schopni na jednoduché úrovni práce s **Canvasem**. Tzn., že dokáží vytvořit plochu, vložit do ní objekt, pohybovat s ním a řešit kolize s dalšími elementy na ploše. Rovněž by měli dokázat pracovat s rozvržením obrazovky tak, aby byla responzivní pro různá rozlišení obrazovky. V neposlední řadě by měli být schopni využívat hodnoty ze senzoru orientace, případně na základě dat získaných z metody `.Flung` objektů umístěných na ploše **Canvas**.

V algoritmickém světě by mělo být po dokončení lekce žákům jasné, co je to podmínka a co jsou to funkce, případně procedury. Žáci by měli být schopni pracovat s logickými hodnotami potřebnými k implementaci podmínek.

14.2.7 Námět na domácí úkol

Hru je možné rozšířit o další překážky. Kromě prostého balónu mohou mít i podobu obrázku (při využití prvku **ImageSprite**). Totéž platí i o cíli. Toto rozšíření by potom bylo možné dokončit v rámci domácí úlohy.

14.2.8 Možné potíže

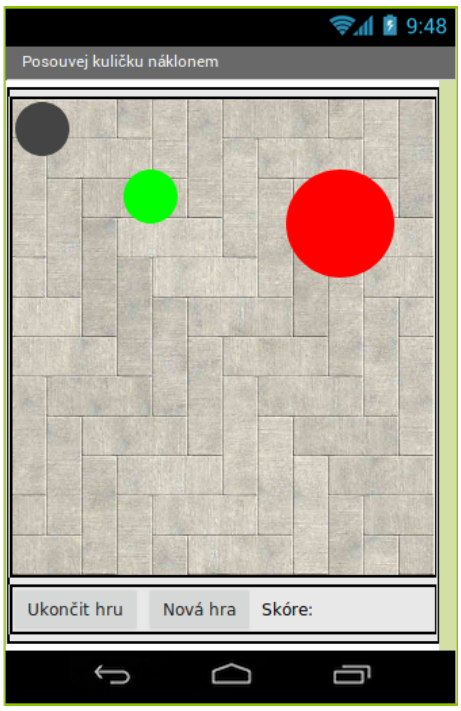
Žáci mohou narazit na stejné potíže zmíněné i ve výše uvedeném bloku. Kromě toho z rostoucí složitostí plyne dále také:

- Žáci mohou mít potíže s pochopením principů buďto algoritmických prvků, nebo proč se určité věci v MIT App Inventoru vytvářejí určitým způsobem. V takovém případě je patrně jediným vhodným řešením zpomalit, byť za cenu toho, že úlohu nemusejí žáci stihnout dokončit včas. Pokud tato situace nastane, je vhodné věnovat se pomalejším žákům zvlášť

a rychlejší žáky neomezovat a nechat pokračovat v úloze dále. Mohou například plnit části z námětu na možnou domácí práci, popřípadě nechat je volným stylem úlohu dále rozšiřovat, napadne-li je vlastní možné rozšíření.

- Při počítání skóre bez použití proměnné, tj. přímo se získáváním hodnoty z prvku **Label**, je nutno inicializovat tento prvek s textovou hodnotou vyhovující číslu. Není tak možné nechat prvek prázdný nebo obsazený řetězcem, protože v takovém případě nedokáže aplikace k výchozí hodnotě přičítat ani odečítat rozdíly skóre. Totéž by platilo i v případě užití proměnné, ovšem v tomto případě již samo prostředí vyžaduje nastavení výchozí hodnoty a nutí tak programátora zamyslet se nad tímto stavem.

14.2.9 Námět na realizaci



Obrázek 3: Ukázka vzhledu aplikace pro pohyb kuličky

Část první K tomu, aby bylo možné na zadání pracovat, je nezbytné být v úvodu ve stavu, kdy je na hlavní obrazovce umístěno libovolné tlačítko. Po kliknutí na něj odkáže aplikace uživatele na novou stránku.

Stránku lze přidat skrze tlačítko „Add Screen . . . “ v horní části obrazovky, přičemž ji lze pojmenovat podle potřeby – v tomto případě například „rollScreen“. To proto, že se na ní bude pracovat s pohybujícím se objektem, konkrétně kuličkou, která se bude kutálet směrem závislým na náklonu zařízení.

Jednotlivé části kódu jsou pak oddělené pro každou obrazovku zvlášť. Proto vše, co se bude v aplikaci odehrávat, musí být programováno v blokovém editoru pro příslušnou obrazovku.

Tlačítku na hlavní obrazovce je dále potřeba přiřadit událost kliknutí, která uživatele po jejím spuštění odkáže na novou obrazovku.

Algoritmus 5 Otevření další obrazovky po kliknutí na tlačítko

```
when baseButton .Click do {
        open another screen screenName "rollScreen"
}
```

Hierarchie grafických prvků nové obrazovky může vypadat například takto:

Algoritmus 6 Možné rozvržení komponent u hry s kuličkou

```
rollScreen {
        VerticalArrangement {
                Canvas (pojmenovaný jako "gameBoard") {
                        Ball (pojmenovaný jako "ball")
                }
                HorizontalArrangement {
                        Button (pojmenovaný jako "exitGameButton",
                           s textem "Ukončit␣hru")
                        Button (pojmenovaný jako "newGameButton", s
                            textem "Nová␣hra")
                        Label (pojmenovaný jako "score", s textem "
                           Skóre:")
                        Label (pojmenovaný jako "gameStatusLabel",
                           bez textu)
                }
        }
}
```

Do obrazovky patří také neviditelný objekt **OrientationSensor**, který lze pojmenovat jako „orientation“.

Objekty **VerticalArrangement**, **HorizontalArrangement** a **Canvas** vyplní plochu horizontálně i vertikálně, čímž budou maximálně přizpůsobeny obrazovce. Vlastnosti vložené instance třídy **Ball** je možné nastavit podle vlastního uvážení. Je však vhodné dbát na to, aby byla dobře vidět.

Jelikož je třeba jednotlivým grafickým elementům přiřadit i logické chování, je nezbytné přepnout se dále do editoru kódu pro obrazovku „rollScreen“. Zde je možné začít nastavením posluchače události `.Click` pro tlačítko „exitGameButton“, který uživatele přesune zpět na hlavní obrazovku aplikace:

Algoritmus 7 Příklad zavření obrazovky po kliknutí na tlačítko

```
when exitGameButton .Click do {
        close screen
}
```

Díky tomu umožní aplikace přepínání se mezi hlavní obrazovkou a obrazovkou hry. Vložený objekt **Ball** je dále potřeba rozpohybovat, což s využitím senzoru orientace může vypadat například takto:

Algoritmus 8 Obsluha změny otočení obrazovky, na jejímž základě je objektu v Canvasu udán směr pohybu

```
when orientation .OrientationChanged do {
        set ball .Heading to otientation .Angle
        set ball .Speed to 5
}
```

V této jednoduché ukázce pojede balónek po obrazovce v závislosti na naklonění zařízení. Zážitek je možné umocnit i tak, že na místo `.Speed 5` bude užit výpočet, který bude rychlost odvíjet od toho, jak moc je zařízení v určitém směru vychýleno:

Algoritmus 9 Změna rychlosti pohybu objektu v Canvasu na základě intenzity náklonu zařízení

```
set ball .Speed to absolute (orientation .Magnitude * 100)
```

Naneštěstí existuje pravděpodobnost, zvláště pracuje-li se s tabletem na místo mobilního telefonu, že příslušný senzor orientace v zařízení chybí. Alternativně je tedy možné na místo senzoru orientace využít vstup z dotykové obrazovky, přičemž k pohybu balónu po obrazovce je pak nezbytné gesto prstem (mírně se podobající výkopu balónu):

Algoritmus 10 Alternativa pohybu objektu po obrazovce jeho odpálením prstem

```
when ball .Flung do {
        set ball .Heading to get heading
        set ball .Speed to get speed
}
```

Jednoduchým užitím podmínky lze také ošetřit, zda je senzor přítomen. V takovém případě preferovat senzor nad prsty:

```
if not orientation .Available { obsah události when ball .Flung do
    }
```

Taková aplikace bude univerzální a bude fungovat v obou prostředích s preferencí senzoru orientace. To rovněž udává námět na případné rozšíření hry o uživatelskou možnost volby, který způsob ovládání preferuje.

Pro větší citlivost (rychlejší a přesnější reakce) je vhodné prvku „ball“ nastavit nižší interval překreslování. Z výchozí hodnoty 100 tak například použít hodnotu 10.

Část druhá Tato část se soustředí především na herní aspekt již vytvořené struktury. Je možné ji plynule spojit s předchozí, případně rozdělit například od další lekce, nezbývá-li dostatek času.

V návrháři prostředí je třeba přidat do aplikace, konkrétně objektu **Canvas**, dna nové objekty typu **Ball**. Jeden bude pro kuličku cílem a druhý pastí.

Algoritmus 11 Možné rozvržení komponent u hry s kuličkou v rámci Canvasu

```
Canvas (pojmenovaný jako "gameBoard") {
        Ball (pojmenovaný jako "ball" - ideálně s vyšší pozicí Z
           než sousedi)
        Ball (pojmenovaný jako "obstacle")
        Ball (pojmenovaný jako "destination")
}
```

Vlastnosti objektů **Ball** mohou být opět různé, je však vhodné dbát na viditelnost, rozpoznatelnost a překryv. Tzn., že by objekty měly být dostatečně velké, kontrastní, s odlišnými barvami a prvek „ball“ by měl mít nastavenu nejvyšší pozici Z.

V editoru kódu se dále žáci seznámí s konceptem funkce, ale i její podmnožiny procedury. Proceduru pro novou hru využijí jak pro zahájení nové hry, neboť ta se bude volat po inicializaci stránky, po kliknutí na tlačítko „Nová hra“, ale i po nárazu pohybující se kuličky do překážky nebo cíle. K tomu, aby byla hra zajímavá, budou vhodné dvě funkce, jejíž návratovou hodnotou bude náhodná pozice X, respektive Y, jejíž hodnoty budou nabývat pozic uvnitř **Canvasu**. Kód pro X může vypadat například takto:

Algoritmus 12 Funkce pro získání náhodné pozice X v rámci Canvasu

```
to getRandomX {
        random integer from (0) to (gameBoard.Width)
}
```

Stejně tak by vypadala i funkce pro získání pozice na ose Y, pouze s tím rozdílem, že by bylo užito hodnoty výšky `gameBoard.Height` na místo šířky.

V proceduře nové hry pak dojde k přesunutí objektů „obstacle“ a „destination“ na náhodné pozice s pomocí volání metody `.MoveTo`, přičemž za X a Y budou dosazena právě volání funkcí k získání náhodných hodnot na příslušné ose:

Algoritmus 13 Funkce „nová hra“, která nastaví pozice prvků na základě náhodných hodnot

```
to newGame {
        call destination .MoveTo {
                x (call getRandomX)
                y (call getRandomY)
        }
        call obstacle .MoveTo {
                x (call getRandomX)
                y (call getRandomY)
        }
}
```

Dále je nezbytné nastavit, co se má stát v případě, že kulička narazí na některý z dalších objektů. V každém případě musí aplikace pozměnit skóre a dále zavolat proceduru pro novou hru.

K tomu ze využije volání `when ball .CollidedWith`, jež se zavolá pokaždé, když „ball“ narazí do nějakého objektu. Logickým rozhodováním lze dále zkontrolovat, s čím kulička právě koliduje, a podle toho spustit příslušnou akci:

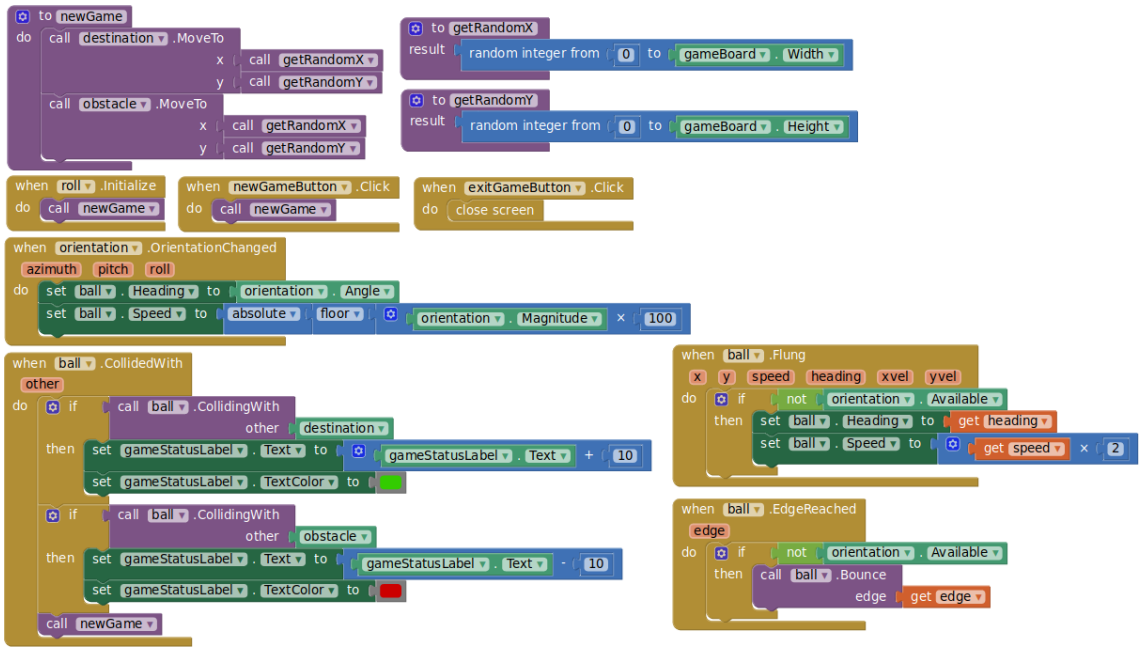
Algoritmus 14 Obsluha události kolize řešící úspěch či neúspěch hráče na základě předmětu kolize

```
when ball .CollidedWith {
        if call ball .CollidingWith {
                other "destination"
                set gameStatusLabel .Text to {
                        gameStatusLabel .Text + 10
                }
        }
        if call ball .CollidingWith {
                other "obstacle"
                set gameStatusLabel .Text to {
                        gameStatusLabel .Text - 10
                }
        }
        call newGame
}
```

Pro ukládání skóre by bylo rovněž možné a do budoucna vhodnější využít proměnnou, ovšem v tomto případě postačí práce přímo s textovým objektem **Label** „gameStatusLabel“.

Zbývá-li čas, je možné hru dále rozšířit například o změnu barvy textu skóre. Tj. v případě, že dojde ke snížení skóre, jej obarvit například do červena, v opačném případě na zeleno. Dále pak přehrát melodii, pokud je skóre menší než 0, a jinou, je-li větší než zvolená hodnota.

Takto vypadá hotová hra i s logikou pro zařízení, která nepodporují senzor orientace:



Obrázek 4: Příklad řešení jednoduché hry v Canvasu

14.3 Hrátky s různými typy proměnných, polem a grafickým seznamem (3. aktivita)

Z předchozí fáze se aplikace nachází ve stavu, kdy na hlavní obrazovce je umístěno pouze prosté tlačítko, jež přesunuje uživatele na zvláštní obrazovku s hrou. Pokud by bylo cílem aplikaci rozšiřovat dále tak, aby pojala i další možné obrazovky, je na místě místo vkládání jednotlivých tlačítek pro každou novou obrazovku s aktivitou vytvořit ucelený a především univerzální seznam, jež bude obsahovat výčet všech možných aktivit.

14.3.1 Vstupní požadavky na žáka

- Žáci mají znalosti potřebné pro předchozí aktivitu.

14.3.2 Obecný cíl

Žák s vedením učitele připraví grafický seznam, naplní ho položkami a dále samostatně na základě stejného principu sestaví algoritmus pro přepnutí na obrazovku odpovídající hodnotě položky seznamu, které se uživatel dotkne.

14.3.3 Konkrétní cíle

- Žák připraví grafický seznam, definuje dvoudimenzionální pole prvků a seznam vhodným převodem propojí s tímto polem.
- Žák vytvoří algoritmus pro přepnutí se na jinou obrazovku na základě zvolené položky seznamu.
- Žák deklaruje proměnnou a nastaví její hodnotu.

14.3.4 Vyučovací zásady

- Zásada trvalosti – Žáci se vracejí k již probranému a opětovným užitím, případně mírným rozšiřováním poznatků, si dovednosti prohlubují a upevňují. Učitel opakuje kostru probírané látky z minulých hodin a dbá na její procvičování.

14.3.5 Postup

Tabulka 7: Průběh aktivity s grafickým seznamem

Čas	**Popis aktivity**	**Vyučovací metody**	**Užité prvky**
5 minut	Úvodní obrazovka aplikace z předchozí aktivity, tj. Sceen1, bude přepracována tak, že z ní zmizí veškerá dosavadní pomocná tlačítka a celou nebo většinu obrazovky vyplní objekt **ListView**.	Instruktáž	ListView
10 minut	Výklad učitele na téma proměnná a datové typy se zaměřením na pole, se kterým bude program dále pracovat.	Slovní metoda, (názorně demonstrační)	–
20 – 25 minut	V blokovém editoru kódu bude založena nová globální proměnná, pojmenovaná například „availableScreens“, jejímž cílem je držet povědomí o všech aktivitách. Tato proměnná se stane předlohou pro seznam, jež z ní bude čerpat hodnoty potřebné pro zobrazování výčtu aktivit. Bude naplněna daty pro existující další obrazovku.	Instruktáž nebo badatelská metoda, (metoda zkušenostního učení, dialogická metoda, názorně demonstrační metody)	–
15 minut	Dále je třeba zajistit, aby po kliknutí na položku v seznamu byl uživatel přenesen ke konkrétní aktivitě. K tomu je možné využít buď přímo metodu seznamu, která se zavolá při výběru prvku v seznamu, nebo pomocného tlačítka, které spustí právě zvýrazněnou položku.	Instruktáž nebo badatelská metoda, (metoda zkušenostního učení, názorně demonstrační metody)	ListView, Volitelně Button
10 minut	Doplnění aplikace o dodatečnou testovací obrazovku obsahující pouze tlačítko pro návrat na hlavní obrazovku a její zanesení do globální proměnné „availableScreens“.	Slovní metoda	Button
5 minut	Závěr hodiny je vhodné vyhradit opakováním vzhledem k tomu, co je to proměnná a pole.	Slovní metoda, (Diskuze)	–

14.3.6 Námět na domácí úkol

Hlavní obrazovku se seznamem je možné dále rozšířit tak, že bude po výběru zvýrazňovat položky odlišnými barvami, ty dále bude možné spouštět dodatečným tlačítkem a seznam uživateli rovněž umožní zrušit výběr položky opětovným kliknutím na jeho jeden prvek.

14.3.7 Výstupy z aktivity

Po dokončení lekce by žáci měli dokázat použít grafický seznam **ListView**, naplnit ho hodnotami a zjistit, která položka seznamu je aktivní, tedy byla vybrána uživatelem aplikace.

Dále by měli rozumět pojmům cyklus a proměnná a být obeznámeni s datovým typem pole.

14.3.8 Možné potíže

Žákům bude činit problém porozumět konceptu proměnné. Nebo mohou nastat problémy s chápáním seznamů, zvláště pak zanořených dvojitých, jež je možné v této úloze využít. Nicméně vzhledem k tomu, že je tato úloha koncipována tak, že v případě roztažení na celých 90 minut vyučovací hodiny dává dostatek času k její realizaci, je možné žákům proměnné i seznamy dostatečně osvětlit a využít diskusí a dalších aktivních metod ke vtažení žáků do tématu.

14.3.9 Námět na realizaci

Hlavní obrazovka může obsahovat prvky dle následující ho návrhu:

Algoritmus 15 Možné rozmístění komponent pro ukázku použití grafického seznamu

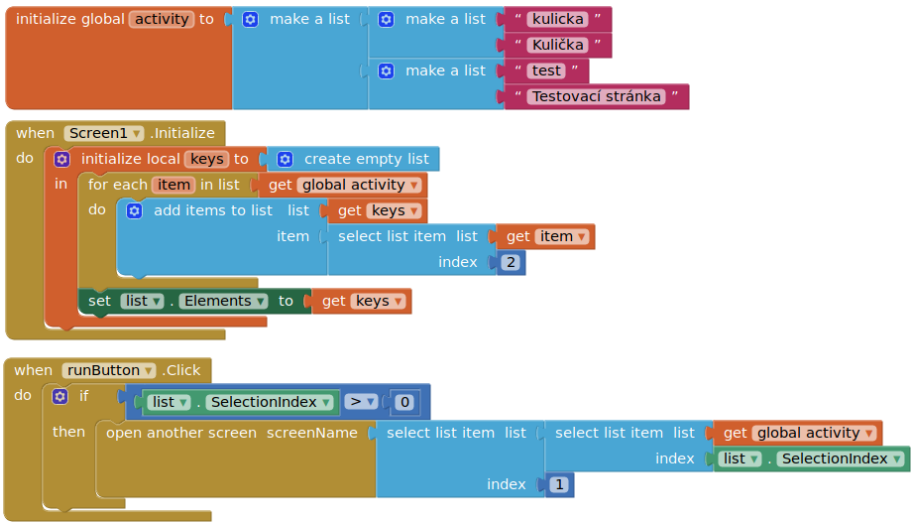
```
Screen1 (volitelně lze skrýt title odškrtnutím položky TitleVisible
    ) {
        Label (pojmenované jako "labelHeader", s textem "Zvolte␣hru
            ")
        VerticalScrollArrangement {
                ListView (pojmenované jako "listViewGamePicker")
        }
        HorizontalArrangement {
                Button (pojmenovaný jako "runButton", s textem "
                    Spustit")
        }
}
```

V tomto okamžiku je více než užitečné přiblížit žákům proměnné blíže, vysvětlit jim, co to je a jak se s nimi pracuje, a rovněž začlenit informace o základních datových typech.

V blokovém editoru bude dále definována globalní proměnná, jež bude obsahovat pole s informacemi pro každou obrazovku. To bude sestávat z dalšího vnořeného pole pro každou položku, kde jeho první položka bude vždy název hry a druhá jméno příslušné obrazovky. Pomocí této struktury bude vyplněn seznam a po výběru položky v seznamu a následném kliknutí na tlačítko spuštění nalezne

logika aplikace příslušné jméno obrazovky a pomocí něj spustí danou hru. Při práci s indexy je třeba neopomenout, že MIT App Inventor nezačíná indexovat od 0, nýbrž od 1.

V prostředí bude vytvořena nová testovací obrazovka, která bude obsahovat tlačítko pro návrat na úvodní stranu.



Obrázek 5: Možná realizace grafického seznamu s užitím dvoudimenzionálního pole

Jak z výše uvedeného kódu vyplývá, do pole nazvaného „availableGames“ byly zaneseny celkem dvě položky, každá obsahující klíč a hodnotu. Jedna dvojice pro hru s balónem a druhá pro testovací obrazovku tak, aby bylo možné využít a prakticky si vyzkoušet přepínání mezi více obrazovkami ze seznamu.

14.4 Záznam zvuku s možností přehrávání (4. aktivita)

Zatímco první aktivita se soustředila na tvorbu jednoduché hříčky, zde je cílem vytvořit jednoduchou ale užitečnou aplikaci, která bude ukládat záznamy zvuků do mobilního zařízení, přičemž ideálně rovněž umožní tyto záznamy sama přehrávat.

Protože kontrola existence souboru na disku není v MIT App Inventoru přímo implementována, není do aplikace možné bez použití rozšíření třetích stran[51] přidat kontrolu existence záznamu a tudíž ošetřit všechny možné chybové stavy.

Nejjednodušším způsobem implementace je nechat záznamník zvuku sám uložit audio soubor dle výchozích parametrů. Soubory takto uložené budou mít aplikací definovaný název (závislý na aktuálním datu a čase) a budou umístěny v adresáři `/storage/sdcard0/My Documents/Recordings`.

Pro přehrávač je pak možné zvolit buďto metodu, která dokáže přehrát pouze právě uložený záznam, případně aplikaci rozšířit tak, aby si poslední záznam ukládala do souboru a při opětovném otevření aplikace jej dokázala najít a i starší záznam přehrát. Pro další zvýšení užitečnosti může aplikace ukládat informace o záznamech do databáze a zobrazovat přímo celý seznam všech uložených záznamů, jež bude následně také schopna přehrát. Při tomto řešení je ovšem vhodné umožnit záznamy z databáze taktéž mazat, neboť již déle nemusejí být na úložišti přítomny.

14.4.1 Vstupní požadavky na žáka

- Žáci mají znalosti potřebné pro předchozí aktivitu.

14.4.2 Obecný cíl

Žák samostatně po představení potřebných prvků v prostředí MIT APP Inventor sestaví funkční program pro nahrávání a přehrávání zvuku.

14.4.3 Konkrétní cíle

- Žák zváží použití komponent a vybere ty, které jsou pro danou aplikaci vhodné.
- Žák sestaví algoritmy pro přehrávání a zaznamenávání zvuku.
- Žák průběžně testuje (a odstraňuje chyby na) vznikající aplikaci na mobilním dotykovém zařízení.
- Žák použije textové řetězce a detekuje obsazení proměnné (je-li prázdná či nikoliv).

14.4.4 Vyučovací zásady

- Zásada trvalosti – Žáci se vracejí k již probranému a opětovným užitím, případně mírným rozšiřováním poznatků, si dovednosti prohlubují a upevňují. Učitel opakuje kostru probírané látky z minulých hodin a dbá na její procvičování.

[51] *File Extension* [online]. Uvita (Costa Rica): Pura Vida Apps [cit. 2019-04-04]. Dostupné z: https://puravidaapps.com/file.php.

14.4.5 Postup

Tabulka 8: Průběh aktivity se záznamem zvuku

Čas	**Popis aktivity**	**Vyučovací metody**	**Užité prvky**
15 – 20 minut	Žáci založí novou aktivitu a rozvrhnou si výchozí obrazovku designově podle vlastního uvážení. Musejí ovšem vytvořit tlačítko pro zahájení nahrávání, které bude rovněž nahrávání zastavovat, a k přehrávání záznamu, jež je vhodné umístit na novou obrazovku. Do prostředí dále vloží neviditelné prvky **SoundRecorder** a **Player** (ta se bude nacházet na zvláštní obrazovce, je-li užita)	Slovní metoda, (dialogická metoda, diskuze, metoda zkušenostního učení)	Screen, Button, SoundRecorder, Player
15 – 20 minut	V blokovém editoru kódu dále implementují tlačítko pro zahájení záznamu tak, aby po kliknutí nahrávání začal, nebo ukončil. Příslušně též upraví popisek tlačítka. Rozhodování mezi stavy lze implementovat buď na základě globální proměnné, nebo skrze kontrolu hodnoty textu tlačítka, či pravděpodobně nejlépe na základě kontroly stavu komponenty nahrávání jeho metodami `.StartedRecording` a `.StoppedRecording`.	Problémová metoda, (dialogická metoda)	Button, SoundRecorder, podmínka, volitelně vlastní proměnná
25 minut	Obdobně žáci implementují přehrávací tlačítko, které umožní zahájení přehrávání záznamu i přerušení přehrávání v průběhu. Po dokončení přehrávání je nutné dbát na to, aby tlačítku byl taktéž nastaven výchozí popisek. Zdroj pro audio bude nastaven skrze událost `.AfterSoundRecored` objektu **SoundRecorder**.	Slovní metoda, instruktáž, názorně demonstrační metoda	Button, Player, podmínka, volitelně vlastní proměnná

30 minut	Vhodným rozšířením pro aplikaci je ukládání informace o poslední přehrávané skladbě do souboru. K tomu lze využít jednoduché řešení v podobě úložiště **File**, na němž se po uskutečnění záznamu zavolá metoda pro ukládání textu do souboru o určitém názvu. Název bude stejný jak pro ukládání, tak pro čtení, a jako hodnota bude užit zdroj zvuku (source) na objektu **Player**. Poté již jen postačí se po novém startu aktivity vždy pokusit tento soubor přečíst, a je-li získaná hodnota neprázdná, pak ji nastavit jako zdroj přehrávače. Tím se stane tlačítko pro přehrávání funkční i po opětovném startu aplikace. Takto uložená hodnota se pak přepíše vždy při pořízení nového záznamu.	Instruktáž, (slovní metoda, dialogická metoda, diskuze, problémová metoda, brainstorming)	Button, File a/nebo TinyDB, podmínka
15 minut	Závěr hodiny je vhodné vyhradit rychlému převedení vytvořených aplikací ostatním žákům a krátkou reflexí toho, co se v hodině odehrálo.	Diskuze, (slovní metoda)	–

14.4.6 Námět na domácí úkol

Praktickým rozšířením by bylo užití databáze k ukládání povědomí o existujících záznamech. Bylo by tak možné všechny soubory přehrávat i v pozdější době přímo z aplikace. K tomu je třeba implementovat seznam a jeho obsluhu. Rovněž je užitečné umožnit aplikaci záznamy mazat, čímž se zruší jejich zobrazování v aplikaci, ale neodstraní se fyzické soubory na úložišti. Pro správnou funkcionalitu je proto možné využít rozšíření „File Extension“.

Dále je možné rozšířit aplikaci tak, aby umožnila uživateli měnit název souboru zaznamenávaného zvuku. Využije se k tomu vstupní pole TextBox a zde vložená hodnota pak bude předána záznamníku skrze atribut **SoundRecording**.

14.4.7 Výstupy z aktivity

Žáci by měli pochopit, jak fungují prvky **SoundRecorder** a **Player** a dokázat s nimi pracovat. Rovněž by měli mít zkušenost s alespoň jednou metodou práce s perzistentním úložištěm dat, tj. **File** nebo **TinyDB**.

14.4.8 Možné potíže

V průběhu hodiny budou žáci věnovat velké množství času testování. Budou sami vydávat nejrůznější zvuky jen proto, aby si je mohli nahrát a následně je budou přehrávat. Učitel s tímto musí počítat a dimenzovat výuku tak, aby měli žáci dostatek času i pro vlastní testování.

14.4.9 Námět na realizaci

Prvním krokem je návrh nové obrazovky, přičemž jednotlivé prvky mohou být v jednodušší variantě rozvrženy následovně:

Algoritmus 16 Možné rozmístění komponent aplikace pro záznam zvuku a jeho přehrávání

```
Screen1 {
        VerticalArrangement {
                Button (pojmenovaný jako "startRecordingButton", s
                    textem "Spustit␣nahrávání")
                Button (pojmenovaný jako "playRecordingButtin", s
                    textem "Přehrát")
                Label (pojmenovaný jako "filePathLabel", s prázdným
                     textem)
                Button (pojmenovaný jako "exitActivityButton", s
                    textem "Ukončit␣aktivitu")
        }
}
```

A dále neviditelné objekty **SoundRecorder**, **Player** a eventuálně také úložiště **File**, případně databáze **TinyDB**.

Vhodnější je ovšem aplikaci rozdělit tak, aby komponenty související s přehráváním byly umístěny na samostatné obrazovce.

Následně je třeba přepnout se do editoru kódu a definovat jednotlivým prvkům příslušné funkce. Vhodné je jako obvykle začít definicí toho, co se má stát po stisknutí tlačítka pro ukončení akce. Dále bude nezbytné implementovat základní funkcionalitu záznamníku zvuku. Na událost stisku tlačítka záznamu tak bude zavěšeno volání metody Start a Stop na objektu **SoundRecorder** v závislosti na proměnné nebo textové hodnotě tlačítka, kterou bude logika aplikace měnit. Rovněž na stejném objektu lze využít události `.AfterSoundRecored`, která vrátí tlačítko a případně i proměnnou do původního stavu. Totéž je možné definovat i pro opětovné kliknutí na tlačítko, což ovšem není zcela neprůstřelné, neboť nahrávání se může za určitých okolností přerušit i bez interakce uživatele se zařízením.

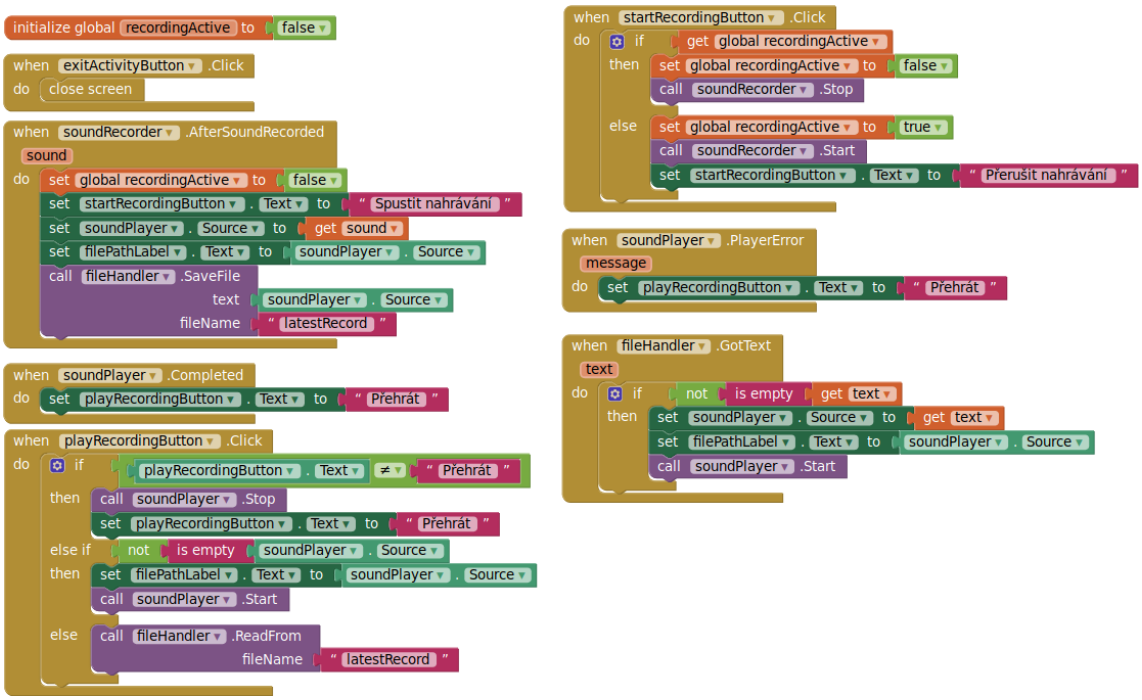
V tomto stavu aktivita dokáže nahrávat zvuky a bez definice konkrétní lokace je ukládá do výchozího adresáře.

K tomu, aby bylo možné záznam přehrát i přímo v aktivitě, využije aplikace objekt **Player**. Na tlačítko pro přehrávání tak bude pověšena událost realizující podobné rozhodování jako v případě záznamníku, která se bude stejným způsobem rozhodovat o tom, zda zvuk přehrát, či ne. A stejným způsobem bude také měnit text tlačítka z „Přehrát“ na „Zastavit“ a naopak.

Na rozdíl od nahrávání zvuku je v tomto případě vhodné měnit stav tlačítka pro přehrávání hned na dvou místech a to v metodě `.Completed` objektu **Player**, jež se zavolá vždy po dokončení přehrávání nahrávky, ale také v metodě `.PlayerError` stejného objektu, neboť nahrávka může být poškozena či kompletně v úložišti chybět a aplikace by měla s touto situací počítat.

Na závěr je možné úlohu rozšířit o perzistentní úložiště, které si bude vždy pamatovat umístění poslední nahrávky a tu pak po opětovném otevření aplikace dokáže načíst tak, že ji bude možné

přehrát. Nejjednodušším řešením je využít ukládání cesty k nahrávce do souboru s pomocí objektu **File** a jeho metody `.SaveFile`. Název souboru pro ukládání této informace bude stejný též pro čtení metodou `.ReadFrom`. Po načtení textu v metodě `.GotText` objektu **File** lze dále verifikovat, že tato hodnota není prázdná, což by indikovalo, že příslušný soubor v zařízení ještě nebyl vytvořen nebo je z nějakého důvodu prázdný. Pokud ovšem získaná hodnota není prázdná, bude nastavena jako zdroj přehrávače **Player** a příslušná nahrávka spuštěna.



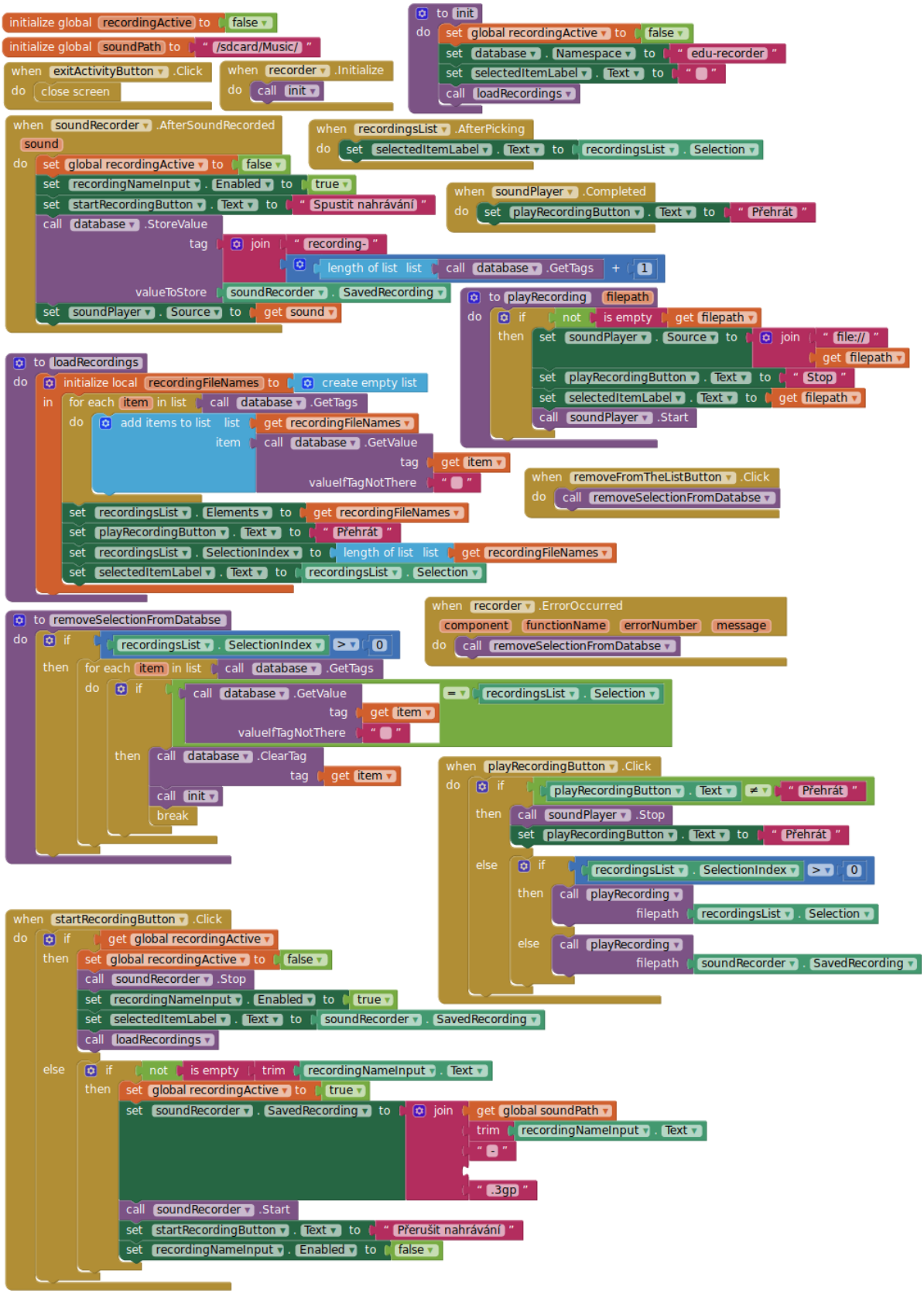
Obrázek 6: Jednodušší varianta nástroje pro záznam zvuku

Námětem na rozšíření a lepší řešení je použít na místo úložiště **File** objekt databáze **TinyDB**. To především proto, aby bylo možné z aktivity přehrávat i starší záznamy a ta měla povědomí o všech již dříve nahraných záznamech. Ačkoliv i do souboru je možné strukturováním vkládat další záznamy, jejich odstraňování by bylo obtížnější. V databázi lze poměrně snadno realizovat oboje.

Do obrazovky tak bude přidán objekt **ListView**, jehož obsahem budou z databáze získané cesty k nahrávkám. Do ní se budou vkládat vždy po dokončení záznamu.

Tlačítko pro přehrávání pak musí být upraveno tak, že bude přehrávat právě zvolenou nahrávku, tedy jako zdroj přehrávače použije hodnotu vybrané položky z **ListView**.

Přibude rovněž tlačítko k mazání vybraného záznamu z databáze, neboť uživatel může takovouto akci požadovat např. proto, že z úložiště byl již tento soubor odstraněn. Naneštěstí MIT App Inventor nedokáže bez dodatečného rozšíření třetích stran pracovat přímo se soubory, a tak není možné jejich setrvání na disku ověřovat, ani je přímo z aplikace mazat. Jedná se tedy o prosté odstranění vazby aplikace na příslušné soubory, nikoliv o práci přímo s nimi.



Obrázek 7: Pokročilejší řešení záznamníku zvuku

14.5 Hra s raketou (5. aktivita)

V této hře je cílem doletět s raketou od spodního okraje displeje po jeho horní okraj. Hráč bude mít omezenou kapacitu paliva a bude muset překonávat gravitaci i odpor vzduchu. K ovládání poslouží jediné tlačítko a to pro zážeh, jehož spuštěním dojde k zažehnutí motorů a zrychlení rakety směrem vzhůru. Pokud palivo dojde nebo není-li tlačítko zmáčknuto včas, začne raketa padat směrem dolů. Nárazem do spodní hrany obrazovky hra končí prohrou, naopak doletem až k hornímu okraji končí výhrou.

14.5.1 Vstupní požadavky na žáka

- Žáci mají znalosti potřebné pro předchozí aktivitu.

14.5.2 Obecný cíl

Žák v závislosti na úvodních informacích navrhne a otestuje hru, jejíž podstatou je let rakety od spodního okraje obrazovky k hornímu a s ní související nezbytná detekce její pozice

14.5.3 Konkrétní cíle

- Žák zváží použití komponent a vybere ty, které jsou pro danou aplikaci vhodné.
- Žák sestaví komponenty včetně jejich obrázků na pozadí tak, aby jejich vlastnosti budily dojem hry s raketou.
- Žák ve spojitosti s komponentou **Clock** aplikuje algoritmus, který zajistí plynulý pohyb objektu, včetně simulace gravitace.
- Žák navrhne algoritmus pro detekci kolize rakety s horní či spodní hranou **Canvasu**.

14.5.4 Vyučovací zásady

- Zásada trvalosti – Žáci se vrací k již probranému a opětovným užitím, případně mírným rozšiřováním poznatků, si dovednosti prohlubují a upevňují. Učitel opakuje kostru probírané látky z minulých hodin a dbá na její procvičování.

14.5.5 Postup

Tabulka 9: Průběh aktivity s raketou

Čas	**Popis aktivity**	**Vyučovací metody**	**Užité prvky**
20 minut	Žáci si rozvrhnou v návrháři prostředí obrazovku tak, aby obsahovala **Canvas** přes většinu obrazovky, a dále tlačítko pro zahájení hry a opuštění obrazovky. V **Canvasu** je rovněž nezbytné umístit prvky **ImageSprite** pro samotnou raketu a odpalovací tlačítko – popřípadě to je možné nahradit standardním tlačítkem v prostředí Androidu. Raketě, tlačítku, ale i **Canvasu** mohou žáci přiřadit vhodné obrázky.	Slovní metoda a diskuze, (dialogická metoda, názorně demonstrační metoda)	Screen, Canvas + Image-Sprite, volitelně Button
20 minut	Žáci vytvoří funkci, jež se bude spouštět vždy při startu nové hry. Ta nastaví parametry rakety (pozici). Dále deklarují příslušné proměnné pro sledování paliva i toho, jestli hra běží. V úvodní funkci je nastaví na výchozí hodnoty.	Slovní metoda, (názorně demonstrační, metoda zkušenostního učení)	Canvas, Image-Sprite, popř. Button
20 minut	Nyní je potřeba raketu „odlepit“ od země. K tomu je vhodné definovat další funkci, jejíž jedno volání obslouží jeden zážeh. Je třeba zkontrolovat, zda je v nádrži dostatek paliva. Pokud je, část paliva odečíst a raketu posunout o kousek vzhůru. Pro plynulý posun rakety lze využít její parametry `Speed` a `Heading`. Tato funkce se zavolá vždy, když se uživatel dotkne tlačítka zážehu.	Slovní metoda, (instruktáž, názorně demonstrační metoda, metoda zkušenostního učení)	Image-Sprite

25 – 30 minut	K tomu, aby měl hráč přehled o dění, je vhodné vytvořit další funkci, jejímž cílem je vypisovat informace o stavu paliva, rychlosti rakety atp. na obrazovku. K využití se vybízí metoda `.DrawText`, kterou lze na **Canvasu** spustit, a s jejíž pomocí lze přímo do herní plochy vypisovat potřebný text. Funkci, která má výpis informací na starosti, je poté potřeba volat všude tam, kde se hodnoty zobrazovaných proměnných mění, tj. např. po stisku odpalovacího tlačítka, při zahájení nové hry nebo později i při změně rychlosti rakety v závislosti na gravitaci.	Slovní metoda, (instruktáž, názorně demonstrační metoda)	Canvas, Button nebo Ball / Image-Sprite
30 minut	Dále je potřeba simulovat gravitaci. Aby byla simulace věrnější, je třeba, aby raketa zpomalovala a po přechodu přes nulovou rychlost začala opět padat. K tomu je již nezbytné použití hodin. Ten v definovaném, ale relativně krátkém intervalu bude kontrolovat rychlost rakety a bude od ní odečítat hodnoty závislé na maximální možné rychlosti v aktuální výšce. Zde je potřeba neopomenout skutečnost, že je nutné kontrolovat, jestli už neletí raketa větší než maximální povolenou rychlostí. V takovém případě hodnotu upravit tak, aby letěla právě jí, čímž dojde k zamezení překročení maximální možné rychlosti.	Instruktáž, (slovní metoda, dialogická metoda, diskuze, názorně demonstrační metoda)	Image-Sprite, Canvas, Clock
20 – 25 minut	Dále musí být dokončena kontrola vítězství a prohry na základě nárazu rakety na horní nebo dolní okraj displeje, pokud hra běží. O stavu je hráče nutné informovat, ideálně výpisem hlášky přes plochu **Canvas**. Po výhře, či prohře je rovněž potřeba hráči umožnit zahájit novou hru.	Slovní metoda, (metoda zkušenostního učení, názorně demonstrační metoda)	Canvas, Image-Sprite
20 minut	Nakonec je ještě nezbytné umožnit hráči držet prst na tlačítku zážehu a nechat raketu letět plynule vzhůru. K tomu bude využit další prvek **Clock** a metody `.TouchDown` a `.TouchUp` zavěšené na dalším tlačítku.	Slovní metoda, (instruktáž, případně badatelská metoda, názorně demonstrační metoda)	Button nebo Ball / Image-Sprite, Clock

20 minut	V závěru hodiny je vhodné nechat žáky předvést své výtvory ostatním a provést jednoduchou reflexi zahrnující opakování toho, co bylo v aktivitě použito, za jakým účelem a čeho bylo dosaženo.	Slovní metoda, Diskuze	–
Doporučení v případě rychlejšího průběhu	Zbude-li čas, je možné věnovat se další dekoraci aplikace. Tedy například umístit do plochy mrak a nechat ho ve spodní části obrazovky poletovat zleva doprava, umístit do plochy hvězdu či více hvězd a ty rovněž na ploše rozpohybovat. Raketě je dále možné přidat efekt zážehu zobrazením plamenů pod raketou.	Diskuze, (dialogická metoda, brainstorming, názorně demonstrační)	Canvas, Ball / Image-Sprite

14.5.6 Námět na domácí úkol

K této úloze je k dispozici obrovské množství možných rozšíření. Například vylepšený výpočet maximální rychlosti, odlišný výpočet při klesání směrem k zemi, zobrazení žáru pod raketou při každém zážehu, rozšířením tlačítka zážehu tak, aby fungovalo i při kontinuálním stisku, lepší logiku efektů (hvězda a mrak), případně doplnění o další. Rovněž by bylo možné přidat do herní plochy překážky a umožnit raketě zatáčet tak, aby se jim mohla vyhýbat.

14.5.7 Výstupy z aktivity

Žáci dokáží pracovat s hodinami pro opakování úlohy. Budou schopni rovněž kromě jednoduchého prvku **Ball** vkládat do **Canvasu** také **ImageSprite**, jež je je nositelem obrázku. Rovněž získají znalosti potřebné pro vypisování textu přímo do plochy **Canvasu**.

Nezbytnou součástí úlohy je práce s hodnotami, ideálně s užitím proměnné. Žáci tak budou schopni využívat matematické funkce, jež v úloze prakticky aplikují. Součástí může být rovněž hledání nejlepšího řešení pro výpočet konkrétních hodnot, tedy úloha spadající do třetí nejvyšší kategorie Hromkovičova dělení vhodného postupu výuky algoritmizace, tj. „Praktická omezení spočitatelnosti“.

14.5.8 Možné potíže

- Protože se jedná o poměrně komplexní a náročnou úlohu, bude žákům velmi pravděpodobně činit problém pochopit jednotlivé vazby mezi prvky. Patrně nejvhodnějším způsobem k tomu, aby měli žáci jasno, je prodiskutovat celou úlohu před jejím začátkem a případně ji i upravit tak, jak ji žáci sami navrhnou – bude-li jejich řešení smysluplné. Vhodným doplňkem je rovněž nákres propojení jednotlivých celků hry.
- Vzhledem k objemu kódu bude žákům chybět místo na obrazovce při skládání bloků. Budou se tak muset velice často posunovat po ploše a nebudou mít přehled o všech zákoutích vlastního kódu. Funkce zoom pravděpodobně také nepomůže, protože při příliš velkém oddálení nebude

text čitelný. Je tak potřeba s tímto z pohledu učitele počítat a dát žákům potřebný čas k tomu, aby se v ploše dokázali zorientovat.

14.5.9 Námět na realizaci

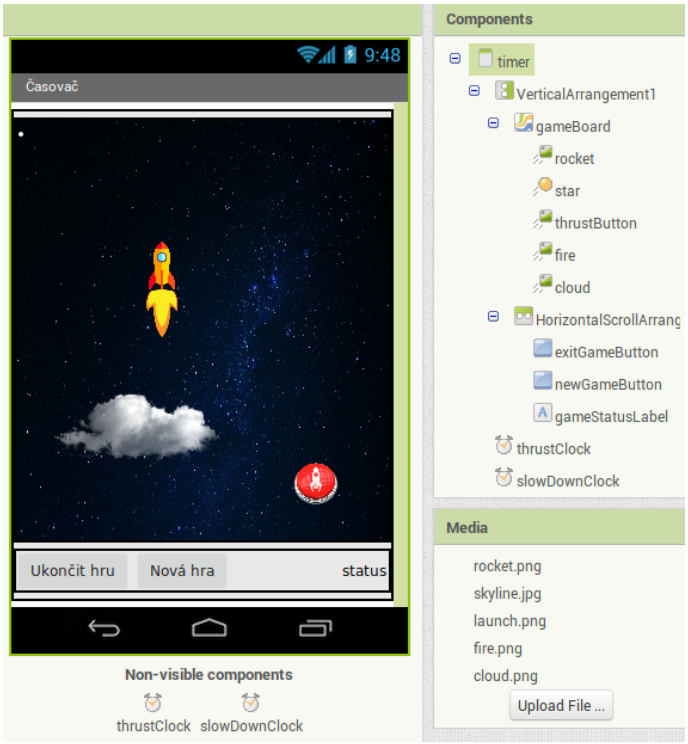
Komponenty na obrazovce mohou být uspořádány následujícím způsobem:

Algoritmus 17 Možné rozmístění komponent u hry s raketou

```
rocketScreen {
        VerticalScrollArrangement {
                Canvas (pojmenovaný jako "gameBoard") {
                        ImageSprite (pojmenovaný jako "rocket", na
                            pozici Z = 2)
                        ImageSprite (pojmenovaný jako "thrustButton
                            ", na pozici Z = 4)
                        Ball (pojmenovaný jako "star", na pozici Z
                            = 1 - dekorační účel)
                        ImageSprite (pojmenovaný jako "cloud", na
                            pozici Z = 3 - dekorační účel)
                }
        }
        HorizontalArrangement {
                Button (pojmenovaný jako "exitGameButton", s textem
                    "Ukončit␣hru")
                Button (pojmenovaný jako "newGameButton", s textem
                    "Nová␣hra")
                Label (pojmenovaný jako "gameStatusLabel")
        }
}
```

Na obrazovce se bude nacházet i neviditelný objekt hodin (**Clock**), jehož náplní bude kontrola rychlosti rakety a postupný útlum v souvislosti s umělou gravitací. Případně zde mohou být umístěny i dvoje hodiny, přičemž druhé umožní kontinuální běh motoru. Pro snazší práci s nimi je vhodné si je příslušně pojmenovat, například: „thrustClock“ a „slowDownClock“.

Jednotlivým grafickým komponentům je třeba přiřadit vhodné obrázky a graficky si rozvrhnout scénu, například tímto způsobem:



Obrázek 8: Možné rozvržení a návrh obrazovky hry s raketou

Objekty „star“ a „cloud“ tvoří pouze dekorační účel a není důležité je ani implementovat ani animovat. Nicméně většinu parametrů jim lze nastavit přímo v grafickém editoru, včetně rychlosti a směru pohybu, což znamená, že v blokovém editoru kódu jim postačí nastavit odraz od stěny a budou i s tak jednoduchými vstupy tvořit živé a poutavé prostředí pro hráče.

Případně je možné při každé nové hře těmto prvkům též nastavit některé hodnoty náhodně a udělat tak hru variabilnější.

Pro hru je vhodné inicializovat následující globální proměnné:

- fuel (stav paliva),
- gameStarted (začala-li hra, bude nastaveno na `true`, v opačném případě `false`),
- thrustHeld (volitelná - logická hodnota, která indikuje, zda je tlačítko zážehu ještě drženo, nebo ne)

a dále mohou být využity následující funkce:

- newGame (uvede hru do výchozího stavu včetně pozic prvků na obrazovce),
- maxRocketSpeed (vrátí maximální rychlost rakety v aktuální výšce),
- thrustAction (obslouží tlačítko zážehu – mj. zvýší rychlost rakety),
- updateStatusLabel (stará se o zobrazení informací o hře v podobě textu na obrazovce).

Při startu nové hry je potřeba se zabývat správným umístěním rakety na obrazovce. Definice výchozích hodnot lze vložit do funkce „newGame“, která bude zavolána vždy po inicializaci okna hry nebo po kliknutí na tlačítko „Nová hra“. Pozici rakety lze měnit její metodou `.MoveTo`, přičemž za X je vhodné dosadit polovinu šířky herního pole a od ní odečíst polovinu šířky rakety. Za Y

postačí dosadit celou výšku herního pole a není potřeba odečítat polovinu výšky rakety, neboť objekty nikdy nepřekročí hranici plochy **Canvasu**. Tak se zajistí, že raketa bude po startu umístěna uprostřed na horizontální ose a dole na vertikální.

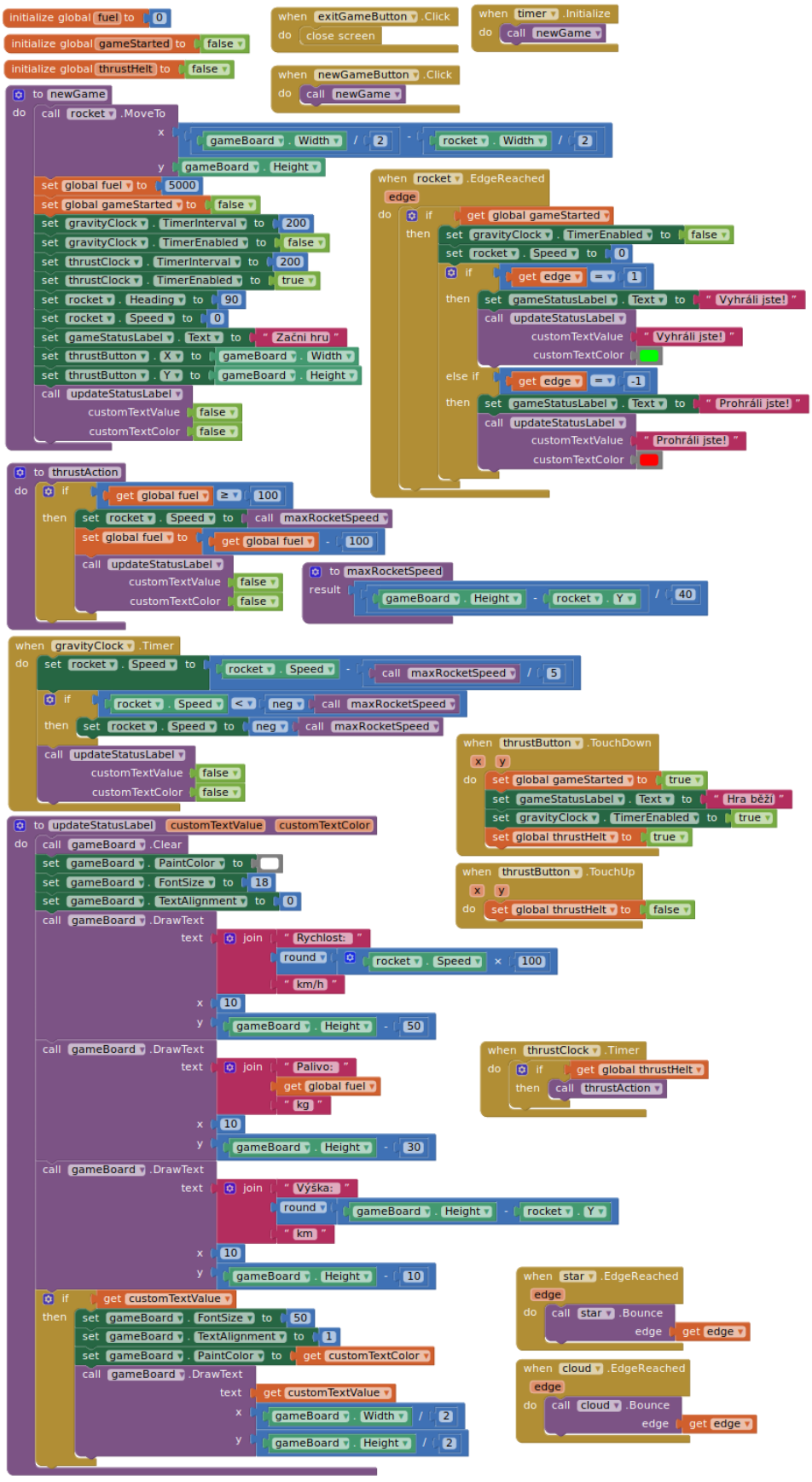
Raketě se rovněž nastaví směr skrze hodnotu `Heading` na 90°, tedy kolmo nahoru, a rovněž rychlost (`Speed`) na 0. Protože se ale `Heading` nemění, je rovněž možné tuto hodnotu nastavit pouze jednou, například přímo v prostředí designéru.

Jednoduchý testovací let pak bude proveden tak, že se na tlačítko „thrustButton“, které je v této hře ideálně součástí **Canvasu**, přiřadí událost `.Touched`. Pro každém zavolání této události aplikace nastaví raketě rychlost (`Speed`) na číslo větší než 0, například 4. Po spuštění hry raketa po kliknutí na tlačítko zážehu poletí směrem vzhůru.

Dalšími kroky je vytvořit funkci pro výpis prvků na obrazovce a funkci pro zjištění maximální rychlosti v konkrétní letové hladině. Dále pak vytvořit funkci pro vzlet a přesunout do ní tělo události `.Touched`. Logiku je vhodné upravit tak, aby zážeh zvýšil rychlost rakety na maximální rychlost v konkrétní výšce.

V neposlední řadě je potřeba zapojit gravitační hodiny, jehož cílem bude zpomalovat raketu a následně ji otočit směrem k zemi v případě, že není dostatečně promptně stisknuto tlačítko zážehu.

Nakonec je možné volitelně rozšířit hru o zážehové hodiny, které budou periodicky volat funkci zážehu po celou dobu stisku tlačítka tak, aby hráč nemusel vyťukávat každý zážeh, zvláště blíže k zemi, kde může být v závislosti na implementaci potřeba mnohem větší síly na překonání vyšší gravitace.



Obrázek 9: Způsob realizace programového aspektu hry s raketou

14.6 Závěrečná ověřovací aktivita (6. aktivita)

K ověření znalostí lze využít jak tvorbu nové aplikace od počátku, tak rozpracované řešení, jejímž cílem je nechat žáky dokončit rozpracovaný zdrojový kód i vzhled aplikace. Zde spadá jak tvorba nových celků kódu, tak případně i oprava špatně navržených částí.

Výhodou druhé možnosti je, že se žáci nebudou muset zabývat všemi aspekty tvorby aplikace v MIT App Inventoru, ale pouze těmi učitelem vybranými. Aktivitu tak lze koncipovat tak, že bude zaměřena především na algoritmické konstrukty a ryzím specifikům prostředí se bude snažit vyhýbat. V tomto směru ovšem záleží na učiteli, co všechno bude chtít ověřovat.

14.6.1 Vstupní požadavky na žáka

Ověření spočívá v testování znalostí, které žáci získali v průběhu plnění předchozích úloh. Žáci tak musí mít již poměrně rozvinuté znalosti práce s MIT App Inventorem k tomu, aby dokázali zadání samostatně vyřešit.

14.6.2 Obecný cíl

Žák opraví základní chyby a optimalizuje úlohu tak, aby byla aplikace ve všech ohledech vyplývajících ze zadání funkční.

14.6.3 Konkrétní cíle

- Žák opraví všechny chyby v aplikaci a opraví je takovým způsobem, aby původně vadné části kódu či prostředí fungovaly a byly přínosné pro celkovou funkčnost aplikace dle zadání.
- Žák najde a přidá další bloky a chybějící části tak, aby byla aplikace plně funkční, tj. splňovala zadání.

14.6.4 Vyučovací zásady

- Zásada trvalosti – Žáci se vracejí k již probranému a opětovným užitím, případně mírným rozšiřováním znalostí, si znalosti osvojují a rozšiřují. Učitel opakuje kostru probírané látky z minulých hodin a dbá na její procvičování.

14.6.5 Postup

Tabulka 10: Průběh závěrečné aktivity

Čas	**Popis aktivity**	**Vyučovací metody**	**Užité prvky**
5 - 10 minut	Žáci si načtou rozpracovanou testovou úlohu. Učitel je obeznámí se zadáním práce.	Slovní metoda	–
30 - 35 minut	Žáci samostatně řeší zadání práce, hledají chyby a dokončují aplikaci tak, aby splňovala zadání.	–	–

14.6.6 Výstupy z aktivity

Žáci by měli samostatně odhalit všechny chyby přítomné v testové úloze. Výsledkem by měla být dokončená a zcela funkční aplikace.

14.6.7 Námět na realizaci

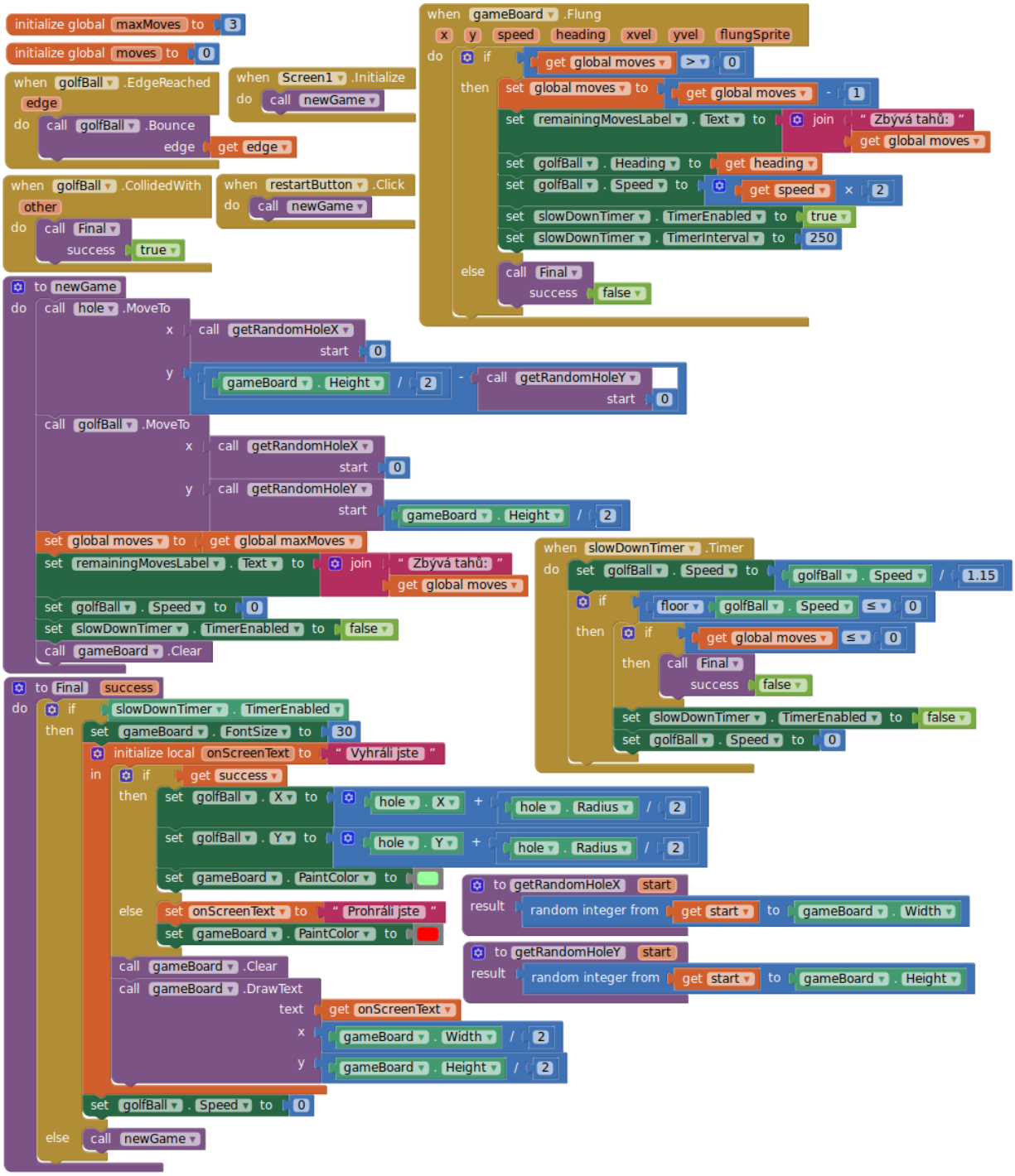
Možným zadáním je dokončení jednoduché hry „golf“, jejímž cílem je trefit se balónem na specifický počet tahů do důlku. Švihem prstu bude balón daným směrem a danou rychlostí odpálen, průběžně se bude zpomalovat a od rohů odrážet. Pádem do důlku hra končí vítězstvím. Naopak vyčerpáním tahů hráč prohrává.

Komponenty na obrazovce mohou být rozmístěny následovně:

Algoritmus 18 Možné rozmístění komponent ověřovací aktivity golf

```
rocketSceeen {
        VerticalScrollArrangement {
                Canvas (pojmenovaný jako "gameBoard") {
                        ImageSprite (pojmenovaný jako "golfBall",
                           na pozici Z = 2)
                        Ball (pojmenovaný jako "hole", na pozici Z
                           = 1)
                }
        }
        HorizontalArrangement {
                Button (pojmenovaný jako "restartButton", s textem
                   "Nová␣hra")
                Label (pojmenovaný jako "remainingMovesLabel")
        }
}
```

Dále je nezbytný prvek hodin, realizující zpomalování balónu, pojmenovaný například jako „slowDownTimer“.



Obrázek 10: Kompletní řešení testové hry minigolf

15 Ověření úloh v praxi

Výzkumný vzorek

Ověřování se účastnilo celkem 13 žáků zapsaných do povinně volitelného předmětu programování. Rozložení třídy bylo různorodé a vyskytovali se zde jak žáci, kteří o programování jevili větší zájem, tak také méně zainteresovaní, spíše neutrálně vystupující jedinci. Dva žáci o předmět nejevili přílišný zájem a bez ohledu na téma plnili aktivity čistě z povinnosti a povětšinou bezmyšlenkovitě opisovali řešení z promítání.

Všichni žáci procházeli standardním vzdělávacím plánem. Ve třídě se nenacházel nikdo s individuálním studijním plánem.

Před vlastním ověřováním se žáci již setkali s některými programovacími aktivitami včetně práce s robotickou stavebnicí LEGO® MINDSTORMS®. Prostředí MIT App Inventoru pro ně však bylo zcela nové a i v porovnání se standardním editorem kódu pro LEGO stavebnice značně odlišné. Na základě úvodních aktivit bylo zjištěno, že žáci nebyli zcela obeznámeni s konstrukty jako jsou proměnné a funkce, neznali koncept objektového programování, metod a přístupu k vlastnostem jednotlivých objektů. Cizí jim byla také představa globální a lokální proměnné nebo datových typů. Potíže měli také při práci s poli.

Vzhledem k možnostem školy bylo pro ověření vlastních úloh a vhodnosti nástroje k rozvoji algoritmického myšlení vybráno Gymnázium a Střední odborná škola dr. Václava Šmejkala v Ústí nad Labem. Ověřování bylo rozděleno celkem do 10 setkání po dvou vyučovacích hodinách, přičemž k tomuto účelu byla k dispozici skupina žáků v povinně volitelném semináři programování.

Reálný průběh aktivit

Testování aktivit probíhalo v pondělních vyučovacích hodinách. Ty byly v rozvrhu umístěny vždy v časech od 12:30 do 14:10. Ve skutečnosti se ovšem nejednalo o standardní dvouhodinovou výuku, neboť ta nebyla přerušena přestávkou a žáci tak končili o délku přestávky dříve.

Tabulka 11: Harmonogram všech ověřovaných aktivit v časové posloupnosti

Pořadí	**Název aktivity / činnost**	**Datum**	**Skutečná délka**
0	Seznámení se s třídou	7. ledna 2019	2 vyučovací hodiny
*	*Školní výlet do Prahy, výuka se nekonala*	*14. ledna 2019*	-
1	Tvorba úvodní aplikace	21. ledna 2019	2 vyučovací hodiny [52]
2	Pohyb kuličky po obrazovce	28. ledna 2019	2 vyučovací hodiny
3	Hrátky s různými typy proměnných, polem a grafickým seznamem	4. února 2019 a 11. února 2019	3 vyučovací hodiny[53]
4	Záznam zvuku s možností přehrávání	18. února 2019 a 25. února 2019	3 vyučovací hodiny[54]
5	Hra s raketou	4. března 2019	2 vyučovací hodiny
*	*Jarní prázdniny*	*11. března 2019*	-
5	Hra s raketou	18. března 2019	+ 2 vyučovací hodiny
6	Závěrečná ověřovací úloha	25. března 2019	1 vyučovací hodina

Výuku zajišťoval výhradně autor práce s dohledem standardního vyučujícího, který figuroval jako pozorovatel a průběh hodin zaznamenával.

Příprava

Příprava předcházející realizaci akčního výzkumu spočívala v návrhu úloh a tvorbě podkladů pro výuku, stejně jako záznamových archů pro pozorovatele i dotazníků pro žáky, viz přílohy na stranách 121 a 122.

Úlohy byly navrženy na základě požadovaných cílů tak, aby zahrnovaly všechny základní algoritmické konstrukty, ale věnovaly se rovněž důležitým, především potom komplexnějším a hůře uchopitelným prvkům MIT App Inventoru, které jsou potřebné pro vytváření velkého množství aplikací. Na tomto základě bylo připraveno 5 samostatných úloh a jedna ověřovací aktivita, která shrnuje nové znalosti a vybírá z nich některé podstatné aspekty práce v MIT App Inventoru.

Dále byly sestaveny podklady pro žáky samotné a to v několika formách. Nejprve byly vytvořeny náměty na realizaci, uvedené u každé z úloh níže. Kromě toho byl vytvořen kurz Moodle (viz příloha na straně 120), který byl tematicky rozdělen do šesti sekcí podle probíraných témat. Součástí tohoto kurzu byla jednoduchá zadání jednotlivých úloh s konkrétními požadavky, jež mají výsledné aplikace či hry splňovat. Rovněž byly v kurzu přítomny soupisy konkrétních postupů, které specifikovaly konkrétní kroky a jejich pořadí, ve kterém se mají pro splnění zadání a dokončení úlohy provádět.

[52] Aktivita se vzhledem k technickým potížím vlekla a trvala déle, než měla. Čistá práce na úloze nicméně odpovídala časovým požadavkům zadání.
[53] Jedna vyučovací hodina byla věnována administračním záležitostem stálého vyučujícího.
[54] Aktivita byla doplněna o práci s databází, která byla vnořena do zbývajícího času výuky.

15.1 Úvodní aplikace (1. aktivita)

15.1.1 Průběh

Na začátku hodiny byl žákům sdělen cíl aktivity, který spočíval v seznámení se s prostředím a tvorbou jednoduché aplikace. Ta byla žákům představena funkční ukázkou na mobilním telefonu, jež rovněž sloužila jako motivační prvek.

Dále byli žáci stručně seznámeni s prostředím MIT App Inventuru, zvláště pak způsobem, jak si vytvoří nový projekt a jak sestaví aplikaci z funkčních celků, tedy z pohledu návrhu prostředí stejně jako i z pohledu skládání bloků kódu. Představení vývojového prostředí proběhlo krátce, avšak dostatečně informativně a z tohoto pohledu žádné podněty ze strany žáků nezazněly.

Při realizaci úloh v praxi nastaly již v samém počátku komplikace spojené s restrikcemi aktivit, které je možné na školní WiFi síti s žákovskými účty provozovat. Vzhledem k tomu, že tato omezení nebylo možné v souladu s konsenzem školy vypnout, a vzhledem k tomu, že ani software pro propojení obou zařízení skrze USB nepracoval správně, byl průběh první hodiny značně narušen.

Všechny problémy lze shrnout následovně:

- Potíže s konektivitou.
- Část žáků využívala mobilní připojení, část ovšem tuto možnost neměla k dispozici.
- Ti, kteří měli mobilní připojení, měli dostatek času zkoušet vytvářet aplikaci dle vlastní fantazie a několik žáků se tak z vlastní iniciativy posunovalo poměrně rychle kupředu.
- Část žáků naopak zůstávala pozadu a to jednak proto, že vzhledem ke komplikacím s připojením nevěnoval učitel dostatek pozornosti práci na aplikaci, tak i proto, že sami nedokázali aplikaci bez vlastního testování vyvíjet. Bylo sice tendencí, aby si mohli tuto aplikaci vyzkoušet na telefonech spolužáků, ale vzhledem k časové náročnosti takového řešení i skutečnosti, že k tomu byla využívána mobilní data konkrétních spolužáků, se nejednalo o nikterak uspokojivé řešení.

15.1.2 Postřehy žáků

Vzhledem k nastalé situaci spočívala velká část průběhu hodiny v samostatné práci jednotlivců. Všichni se ovšem s prostředím na základě úvodního vstupu učitele seznámili na takové úrovni, aby v tomto okamžiku dokázali vytvořit novou aplikaci, vložit do prostředí tlačítko a po kliknutí na něj spustili vlastní sekvenci obsluhy. Žáci například sami přišli na to, že název samotného projektu, stejně jako identifikátorů objektů v prostředí, nemohou obsahovat diakritiku ani jiné speciální znaky, jako jsou mezery.

Je rovněž možné tvrdit, že zatímco pro část třídy bylo prostředí MIT App Inventoru zatím poměrně vzdálené, některým se podařilo zcela bez podnětů učitele pracovat rovněž na komplexnějších programech, ačkoliv největší zájem budily prvky v režimu návrhu, kde žáci užívali především obrázků ke grafickému oživení aplikace.

Přítomen byl i žák, který vytvořil projekt, jež takřka splňoval původní zadání a implementoval tlačítko, přehrávání zvukového souboru a nad rámec na základě komplexní odpovědi učitele na dotaz, který mu žák v této souvislosti položil, využil dokonce hodiny, s jejichž použitím byla po uplynutí nastaveného času spuštěna akce pro schování obrázku.

15.1.3 Dotazníky

Žáci s aktivitou neměli žádný zásadní problém a seznámení s prostředím proběhlo velmi snadno. Nebýt potíží se špatně nakonfigurovanou sítí WiFi, bylo by možné s přehledem stihnout definovaný čas a s velkou pravděpodobností také plošně zařadit i další rozšiřující úlohy nebo přejít k další úloze v řadě.

Tabulka 12: Anonymní žákovské hodnocení (1. aktivita)

porozumění zadání	**náročnost**	**atraktivita tématu**	**poznámka**
1	1	1	
1	1	1	
1	1	2	
1	2	1	„není to psaní kódu“
1	2	2	
2	2	1	„Připojení není domyšlené“
2	2	3	„Hodilo by se lepší připojení na internet“
3	3	2	„moc dobře to nefunguje“
4	3	4	

Stejného dojmu nabyl i pozorovatel výuky, který zaznamenal velké množství dotazů na technické problémy, samotnou aktivitu a její následný průběh však hodnotí pozitivně.

Tabulka 13: Záznam pozorovatele (1. aktivita)

Otázka	**Odpověď**
Na co se žáci ptají?	V hodině se řešily především problémy s připojením, odkud pramenily také otázky. Se samotnou aktivitou žáci problém neměli a ptali se spíše ze zvědavosti. Například: „Jak změním barvu pozadí aplikace na náhodnou?“
Jak moc je potřeba žákům pomáhat? Co zvládnou sami?	Žáci s nástrojem začínali a bylo jim potřeba prostředí demonstrovat a jeho prvky vysvětlit. Po krátké době dokázali s aplikací pracovat i samostatně.
Je postup zvolený pro splnění aktivity vhodný?	Ano
Je vyhrazená časová dotace adekvátní pro splnění úlohy?	S rezervou. Řešení komplikací s nefunkčním připojením mobilní aplikace AI Companion na školní WiFi zabralo příliš mnoho času.

15.1.4 Závěr

V průběhu aktivity se nevyskytly žádné zásadní komplikace s úlohou samotnou, přesto byla výuka značně vleklá a docházelo k citelnému zpoždění s její realizací. Důvodem byly potíže s internetovým připojením, přičemž v tomto směru nepomohla ani promptní instalace ovladačů pro propojení zařízení skrze konektoru USB. Řešením tak nakonec bylo nasadit přídavný WiFi router, připojený do zásuvky Ethernetu přímo ve školní třídě. Tato síť disponovala jinými politikami než školní žákovská síť WiFi. Přes veškeré nesnáze se nicméně během prvního setkání podařilo naplnit všechny úlohou požadované cíle. Bylo využito i prvků, které první úloha uvádí jako nepovinné, a to porovnání a podmínky.

Pro opakované použití těchto aktivit v hodinách, ale i obecně použití MIT App Inventoru při výuce, tak plyne potřeba lepšího otestování kompatibility mobilní aplikace AI Companion v prostředí školní infrastruktury před samotným zahájením výuky. V tomto konkrétním případě sice k testování došlo, ale byla pro něj chybně zvolena učitelská WiFi síť, pro niž se vztahují odlišná pravidla.

Samotná úloha pak byla koncipována zejména jako seznamovací. Sloužila žákům pro vyzkoušení si některých základních prvků prostředí a sestavení jednoduchého kódu. Volný průběh a relativní malá náročnost tak v tomto směru zmenšuje riziko chyby.

15.2 Pohyb kuličky po obrazovce (2. aktivita)

15.2.1 Průběh

Tato druhá aktivita v pořadí byla koncipována jako dvouhodinová, přičemž tento čas akorát stačil k tomu, aby se vše potřebné podařilo i pomalejším žákům dokončit. U rychlejších žáků zbyl také prostor pro diskusi nad vlastními nápady a jejich realizaci. Průběh byl oproti předchozí aktivitě mnohem plynulejší a technické potíže se již takřka nevyskytovaly. Nedocházelo k žádným zásadním komplikacím a ověřování aplikací fungovalo dle představ. Problematickou, ovšem očekávanou stránkou, bylo, že na dvou zařízeních nebyl k dispozici či nepracoval správně senzor orientace a bylo tak nezbytné použít metodu .Flung zmíněnou v návodu.

Část žáků ve třídě se u druhého setkání obměnila. Dva žáci tak neměli reálie potřebné k samostatné činnosti a prostředí MIT App Inventoru jim muselo být stručně představeno. Dalším faktorem byl pozdní příchod žáka. Podporou mu však byly podpůrné materiály, které měli žáci k dispozici jak v prostředí Moodle, tak ve fyzické vytištěné formě. Díky nim tak mohl na projektu začít pracovat od počátku a nebylo nutné se mu věnovat tak důkladně, čímž by o pozornost přicházel zbytek třídy.

Vzhledem k předchozím zkušenostem s programováním bylo překvapením, že většina z žáků neznala pojem funkce. Užitečnost tohoto konceptu ovšem záhy pochopili, neboť použití funkcí bylo nezbytnou součástí úlohy.

15.2.2 Postřehy žáků

Ještě v počáteční části aktivity dokázal jeden z aktivnějších žáků přijít na vylepšení související s pohybem kuličky po obrazovce. Přišel na to, že změnou nastavení obnovovacího intervalu objektu Ball či Sprite z výchozí hodnoty na nižší je pohyb kuličky plynulejší.

Nejen rychlejší žáci měli možnost v hodině vyzkoušet vlastní nápady. Velké části z nich ale bylo nutné s jejich realizací pomáhat. Těm iniciativnějším však mnohdy stačilo jen nastínit možné řešení a zbytek dokázali žáci implementovat sami.

V tomto směru je na místě zmínit, že několik žáků z vlastní iniciativy použilo nejrůznější konstrukty, komponenty a jejich metody, se kterými v této aktivitě ještě nebylo počítáno. Někdo využíval proměnné, další žák skládal barvy z náhodných čísel a dva nejrychlejší si dokonce hráli na základě námětu vyučujícího na možné řešení postupného zpomalování pohybujícího se objektu s prvkem hodin. To vše stíhali realizovat, zatímco zbytek třídy procházel aktivitou standardním způsobem.

15.2.3 Dotazníky

Z dotazníku vyplývá, že se žákům aktivita spíše líbila. Dva zmiňují problémy s připojením mobilního telefonu, které ovšem učitel ani pozorovatel již nezaznamenali a patrně budou souviset s počátečnímu problémy u první aktivity. Ve třídě byli přítomni dva žáci, kteří by raději, než skládání grafických bloků psali přímo kód.

Tabulka 14: Anonymní žákovské hodnocení (2. aktivita)

porozumění zadání	**náročnost**	**atraktivita tématu**	**poznámka**
1	2	1	„radši bych psal kód“
1	2	1	„radši bych psal kód, lepší způsoby downloadu/upgradu“
1	2	1	
1	2	2	„Dalo mi to prostředek k přemýšlení. Chce to zlepšit připojení s mobilem.“
1	2	2	
1	2	2	
1	3	1	„Bavilo mě to“
2	2	1	„zajímavé“
2	2	2	
2	2	3	„COOL“
2	3	1	„Chci toho více“

Jediným technickým problémem, který zaznamenal i pozorovatel, byla nedostupnost senzoru orientace v některých zařízeních, konkrétně u dvou žáků. V jednom případě šlo o klasický mobilní telefon, ve druhém o tablet. Obě tato zařízení sice dokázala detekovat naklonění, ovšem šlo pouze o strohá data potřebná k otočení obrazu, nikoliv k přesné detekci míry náklonu.

U tří žáků dále zaznamenal výskyt problémů s algoritmickými konstrukcemi. Jednalo se konkrétně o prvotní neznalost konstruktu funkce, která byla žákům ovšem vysvětlena a sami si práci s ní na několika případech vyzkoušeli.

Pozorovatel také uvedl, že někteří žáci vynikali a přicházeli na různé způsoby řešení a různá rozšíření svých aplikací, viz postřehy žáků výše.

Tabulka 15: Záznam pozorovatele (2. aktivita)

Otázka	**Odpověď**
Na co se žáci ptají?	Co je to funkce? Proč se mi kulička na obrazovce nepohybuje?
Jak moc je potřeba žákům pomáhat? Co zvládnou sami?	Je patrný rozdíl mezi žáky. Někteří zvládají tvorbu sami a ještě experimentují, jiní chvílemi pouze „opisují“ z projekce
Je postup zvolený pro splnění aktivity vhodný?	Ano
Je vyhrazená časová dotace adekvátní pro splnění úlohy?	Ano

15.2.4 Závěr

Aktivita probíhala dle předpokladů a nepřinesla žádné překvapivé závěry. Se zmíněnými komplikacemi detekce náklonu zařízení bylo počítáno a využilo se v těchto případech doporučeného alternativního postupu. Lze tedy tvrdit, že se aktivita povedla a veškeré cíle byly naplněny. Setkala se vesměs s kladným hodnocením z řad žáků a byla pro ně rovněž motivujícím prvkem, neboť se spoustě z nich podařilo upravit si v základu jednoduchou hříčku k obrazu svému a vyzkoušet si také vlastní nápady a postupy.

15.3 Hrátky s různými typy proměnných, polem a grafickým seznamem (3. aktivita)

Průběh

V této aktivitě byli žáci postaveni před problém, jak implementovat grafický seznam a dále rozhodnout, kterou obrazovku zobrazit na základě zvolené položky seznamu. Ačkoliv s pochopením toho, co je to proměnná problém nebyl, nebyla aktivita ideálně koncipovaná, neboť začínala namísto proměnné s jednoduchým datovým typem rovnou s dvourozměrným polem, což činilo zhruba polovině žákům potíže. Navzdory tomu, že byl tento předpoklad již na začátku zmíněn a s delším vysvětlováním se počítalo, nedařilo se žákům uspokojivě, bez předchozí znalosti alespoň jednodimenzionálního pole problematiku dostatečně osvětlit. Takto koncipované zadání ovšem nebylo vhodné řešit jiným způsobem.

Aktivita ale měla i přes tyto problémy své opodstatnění. Úlohu v současném stavu lze totiž pochopit jako určitou seznamovací vsuvku, jenž má za cíl ukázat žákům především práci s poli a se seznamem, a za tímto účelem byla ve výuce i využita. Posloužila tak jako letmé seznámení se s novými koncepty a k jejich upevnění je vhodné využít aktivitu záznamu zvuku, která je následující v pořadí.

15.3.1 Postřehy žáků

Spíše než s pozitivními reakcemi bylo možné setkat se s problémy žáků vyplývajícími z náročnějšího zadání. Žáci tak tápali především při práci s dvoudimenzionálním polem a dotazovali se zvláště tě na to, jak celý koncept práce s ním funguje. Měli problém rozlišit prvky pole a uvědomit si, kam se jednotlivé bloky kódu odkazují, tj. které položky z pole poptávají. Obecně tak hodnotili úlohu jako náročnou a méně zábavnou než předchozí aktivitu s kuličkou.

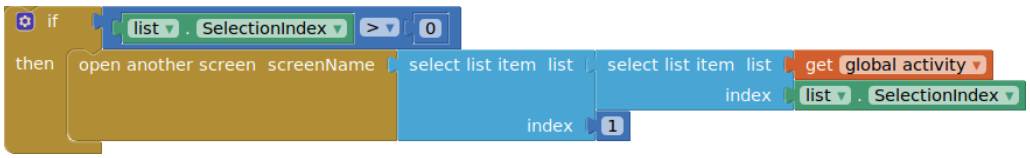
15.3.2 Dotazníky

U této aktivity je z dotazníku na první pohled patrná menší obliba u žáků, stejně jako poměrně velká náročnost. Je to dáno tím, že tato úloha nespočívá ve tvorbě žádné hry ani jiné zábavné aktivity, a naopak se soustředí čistě na problém proměnných a polí.

Tabulka 16: Anonymní žákovské hodnocení (3. aktivita)

porozumění zadání	náročnost	atraktivita tématu	poznámka
1	1	1	
1	4	1	„Bylo to super“
2	1	3	
2	2	3	
2	2	4	
2	3	2	„Dobrý“
2	3	2	„Moc jsem nevěděl, co se děje, ale splnil jsem.“
3	3	2	
3	2	3	
3–4	3–4	3	„Nic moc“
4	3	4	„Byl jsem mimo“

To vyplývá také z hodnocení pozorovatele, který tyto obtíže reflektoval. Problémy se objevovaly také u aktivnějších žáků a zdaleka nejvíce žáci tápali při snaze porozumět následujícímu bloku kódu:



Obrázek 11: Problémová část práce s polem

Konkrétně měli žáci největší potíže rozeznat, která položka v poli je metodě `.open another screen (screenName)` předávána.

To potvrzuje také vyučující, který tyto problémy rovněž zaznamenal.

Tabulka 17: Záznam pozorovatele (3. aktivita)

Otázka	Odpověď
Na co se žáci ptají?	Co je to proměnná? Co je to pole? Jak se přistupuje k položce pole?
Jak moc je potřeba žákům pomáhat? Co zvládnou sami?	Žáci vyžadují větší množství pomoci. Sami zvládnou proměnnou inicializovat, vyplnit hodnotami, ale již ne k položkám v poli správně přistupovat.
Je postup zvolený pro splnění aktivity vhodný?	Bylo by vhodnější postupovat pomaleji a nejprve se zaobírat proměnnou s jednoduchým datovým typem a až později poli.
Je vyhrazená časová dotace adekvátní pro splnění úlohy?	Ano

15.3.3 Závěr

Cíle aktivity se navzdory potížím z velké části splnit povedlo. Užití proměnných žáci zvládli, rovněž pochopili, proč bylo využito dvourozměrného pole. Většina žáků nicméně nedokázala pracovat

s výběrem prvků z pole podle toho, zda je cílem nalézt klíč nebo hodnotu na n-té pozici ve dvoudimenzionálním poli. Řešení úlohy pak spočívalo přinejmenším v tomto aspektu ve strohém opisování z projekce.

Vzhledem k vyšší náročnosti a velmi nízké atraktivitě by bylo v tomto stádiu mnohem vhodnější užít jednodušší úlohu, která by se na místo takto komplikovaného případu zabývala nejprve čistě proměnnými s jednoduchými datovými typy (text, číslo) a později se případně zabývala stroze jednorozměrnými poli. Některé části by bylo rovněž možné předsunout do předcházející úlohy, kde by bylo užití proměnné vhodné k ukládání stavu skóre. Přinejmenším by bylo nezbytné upravit stávající úlohu tak, aby alespoň v prvním návrhu pracovala s polem jednodimenzionálním, na místo komplikovanějšího pole dvoudimenzionálního.

15.4 Záznam zvuku s možností přehrávání (4. aktivita)

15.4.1 Průběh

Zatímco v předchozí aktivitě nebyl příliš velký prostor pro vlastní nápady, práce se zvukem byla patrně i z tohoto důvodu žáky vnímána mnohem lépe. Již od počátku byla úloha dobře uchopitelná a v momentě, kdy dokázala aplikace zvuky nahrávat a následně i přehrávat, začala být pro žáky evidentně zajímavá. V průběhu práce na úloze si tak žáci zkoušeli nahrávat různé zvuky a při tom své dílo dále vylepšovali.

Aktivita začínala dvěma jednoduchými obrazovkami, které se starají o nahrávání a přehrávání zvuků. Následným rozšířením bylo použití souborového úložiště tak, aby bylo umožněno přehrávat poslední nahraný zvuk i po ukončení aplikace.

Nad rámec původního zadání se na základě žádosti druhého učitele pokračovalo v rozšiřování aplikace dále. Do aplikace byl zakomponován grafický seznam, jež obsahoval celou historii nahrávek. K tomuto účelu byla využita na místo původního souborového úložiště databáze. V této fázi žáci opět narazili na nutnost použít grafický seznam v kombinaci s hodnotami z pole. Přičemž si vyzkoušeli také pokročilejší práci s textem, která spočívala v úpravě řetězce do podoby vhodné k zobrazení (extrakci jména souboru s příponou z plné cesty).

Na rozdíl od předchozí úlohy se již žákům s polem (extrahovaným z databáze) i grafickým seznamem dařilo úspěšně pracovat. Někteří ovšem nestíhali a zatímco výklad probíhal dále, ještě testovali své aplikace. Mezi důvody patřily zejména opravy chyb, které v aplikaci měli, případně i pomalejší internetové připojení. Učitel tak musel reagovat na individuální dotazy týkající se nesprávné funkčností aplikací žáků, přičemž ve všech případech se jednalo o chyby z nepozornosti, které byly učitelem během výkladu zmíněny.

15.4.2 Dotazníky

Naneštěstí výsledky ankety neodpovídají aktivitě žáků v průběhu řešení. Zvláště pak položka „porozumění zadání“.

Tabulka 18: Anonymní žákovské hodnocení (4. aktivita)

porozumění zadání	**náročnost**	**atraktivita tématu**	**poznámka**
1	5	2	
2	2	2	
2	4	3	
2	4	4	
3	3	1	Bylo to celkem zajímavé a dozvěděl jsem se opět něco nového
3–4	3	2	Dobrý
4	2	3	Nebaví mě moc pracovat v MIT, protože rozhraní je nedokonalé, možnosti jsou redukovány a emulátor (a celkově vše) obsahuje mnoho bugů. Osobně bych raději pracoval v Xamarinu.
4	3	1	
5	2	1	

Pozorovatel ve svém dotazníku potvrzuje skutečnost, že se někteří žáci ptali na již vysvětlené věci, neboť se ve svých aplikacích zdrželi a nestíhali výklad. Konkrétně uvádí čtyři takové případy v průběhu celé výuky.

Tabulka 19: Záznam pozorovatele (4. aktivita)

Otázka	**Odpověď**
Na co se žáci ptají?	Proč mi to nefunguje? Kde je chyba?
Jak moc je potřeba žákům pomáhat? Co zvládnou sami?	Žáci poměrně dobře tuší, co mají dělat. Prostředí MIT App Inventoru jim už není tak neznámé a většinu zvládají sami. Obtíže mají při práci s poli a grafickým seznamem, ačkoliv již mnohem méně fundamentální než u předchozí aktivity.
Je postup zvolený pro splnění aktivity vhodný?	Ano
Je vyhrazená časová dotace adekvátní pro splnění úlohy?	Časová dispozice je v pořádku. Zadání nepočítá s rozšířením o databázi.

15.4.3 Závěr

Cíle aktivity se podařilo u všech žáků bez problémů naplnit. Všichni žáci rovněž porozuměli, proč se konkrétní postupy používají. Na druhou stranu průběh aktivity mohl být buďto striktnější tak, aby žáci procházeli tématem rychleji, nebo spíše ze strany učitele volnější tak, aby jim ponechal dostatek času pro vhled do problému. Bylo by tak vhodné úlohu protáhnout alespoň na celé tři vyučovací hodiny, případně na čtyři se zakomponováním grafického seznamu a databáze, jež by byly vhodným doplněním zadání.

15.5 Hra s raketou (5. aktivita)

15.5.1 Průběh

Touto aktivitou se žáci vrátili zpět ke Canvasu, ovšem v rozšířené podobě se zaměřením na jeho další možnosti. Žáky z počátku velmi bavilo navrhnout si prostředí hry podle sebe. Nechyběla ani fantazie, neboť se ve třídě objevovaly různé obrázky raket včetně jedné tenisové.

Ačkoliv práce na aktivitě, ač komplexní, probíhala v podstatě hladce, vyskytlo se v počátečních fázích i několik potíží. Prvním problémem bylo samovolné posouvání rakety k levému okraji obrazovky ihned po startu hry, což nesouviselo přímo s danou aktivitou, ale zavinila to chyba učitele a opomenutí nastavení směru pohybu rakety na vertikální při startu hry. Řešení se část žáků aktivně účastnila a snažila se rovněž přijít na to, co je v aplikaci špatně. Zpoždění, které tento problém přinesl, trvalo asi 5 minut. Komplikace se vyskytovaly i u hodin pro zpomalování a následný pád rakety. Ty ale byly ve větší míře spojeny s opisováním žáků řešení z tabule, navíc chybného, na místo vlastní implementace, která měla vycházet ze společného brainstormingu, který sekci předcházel.

Kromě algoritmických potíží se vyskytoval dále také problém s přetékáním obrazu mimo obrazovku, kdy se plocha Canvas roztahovala více, než kolik by měla. Jedná se o kombinaci problémů se živým zobrazováním aplikace v ověřovacím módu, kdy restart aplikace ve většině případů problém vyřeší, případně o chybu v nastavených jednotkách. V několika málo případech byla chyba velmi perzistentní a pro její vyřešení bylo potřeba zcela změnit nastavení Canvasu tak, aby plochu vyplňoval procentuálně. Příčinu tohoto chování se nepodařilo zjistit, neboť se i stejná aplikace chovala odlišně na různých zařízeních.

15.5.2 Postřehy žáků

Hra byla koncipována tak, aby při zážehu došlo ke zvýšení rychlosti na její maximální hodnotu. Díky žákovskému přičinění (jejich věcnému podnětu) však došlo k úpravě rychlosti rakety tak, aby byla k její aktuální rychlosti přičtena zvolená předem definovaná hodnota a raketa tak skokově i ze záporných hodnot nezískala maximální rychlost.

V původním návrhu hry s raketou byla rovněž otevřena možnost vizualizovat v průběhu zážehu plameny u trysek. K tomuto nebylo při výuce dostatečné množství času, přesto jeden žák zcela sám, bez zmínění této možnosti, vizualizaci realizoval.

Rovněž žák, který se s v první části výuky nedostavil, dokázal v dalším setkání samostatně hru s raketou sestavit (při opomenutí některých částí jako je výpis stavu paliva, rychlosti rakety či detekce hranice obrazovky), přičemž pro implementaci volil vlastní postupy. Z nich stojí za povšimnutí zejména specifické řešení zrychlování rakety, kde přičítaná rychlost měla několik stupňů a odvíjela se od aktuální rychlosti rakety. Rovněž v době zážehu nebyly aktivní bloky, realizující zpomalování rakety gravitací.

15.5.3 Dotazníky

Z hodnocení žáků vyplývá, že se jim aktivita líbila a námět je zaujal. Přesto se až na dva žáky mezi vyplněnými dotazníky nevyskytlo žádné upřesnění hodnocení v podobě poznámky. Z uvedených

poznámek pak byla jedna negativní, čemuž odpovídalo i číselné hodnocení aktivity. Důvodem je však spíše než atraktivita tématu nelibost samotného nástroje MIT App Inventor, kterým se již žák nechce dále zabývat. Vzhledem k tomu, že se ale jednalo o poslední aktivitu v tomto tématu, na kterou již navazuje pouze ověřovací úloha, je žákovu výtku možné považovat za vhodně načasovanou.

Autor jednoho dotazníku dále využil i hodnocení, které neodpovídalo standardní škále. Žák tím tak dal najevo, že mu bylo téma úkolu velmi blízké a zaujalo ho. Je však zvláštní, že měl současně potíže s porozuměním zadání.

Tabulka 20: Anonymní žákovské hodnocení (5. aktivita)

porozumění zadání	**náročnost**	**atraktivita tématu**	**poznámka**
1	1	1	
1	2	1	
1	2	1	
1	2	1	
1	3	1	
2	2	1	
2	3	2	
3	1	0,5	
3	2	1	
3	2	2	celkem pohodička
3	3	4	Už mě to moc nebaví, začal bych s něčím jiným

Podle pozorovatele bylo u této aktivity největším problémem správně navrhnout a naprogramovat obsluhu zážehu a následné klesání rakety. K tomu bylo potřeba provést poměrně velké množství kroků, které ovšem většina žáků sama nerealizovala a řešení spíše opisovala z projekce. Objevovalo se tak větší množství chyb a raketa se z počátku nechovala tak, jak by měla. Pouze pár lidí postupovalo podle vlastního uvážení - u nich ovšem musel učitel řešit jiné problémy, na které sami naráželi. Dále byly zmíněny rovněž problémy s posuvem rakety k okraji displeje po startu hry a přetékáním obrazu mimo hranice obrazovky.

Pozorující učitel na druhou stranu detekoval také zvýšený zájem žáků nejen o řešení zadané úlohy, ale všiml si i dvou žáků, kteří pracovali v prostředí MIT App Inventoru mimo školu (přes jarní prázdniny) a během výuky učiteli své aplikace ukázali.

Tabulka 21: Záznam pozorovatele (5. aktivita)

Otázka	**Odpověď**
Na co se žáci ptají?	Proč se mi raketa posouvá? Proč mi přetéká obraz mimo obrazovku? Co dělám špatně?
Jak moc je potřeba žákům pomáhat? Co zvládnou sami?	I pomalejší žáci se při tvorbě aplikace dokáží v problematice orientovat. Některé složitější algoritmické konstrukce jim ale činí potíže a ty pak buď kontrolují, nebo častěji opisují z tabule.
Je postup zvolený pro splnění aktivity vhodný?	Ano
Je vyhrazená časová dotace adekvátní pro splnění úlohy?	Ano

15.5.4 Závěr

Cíle aktivity se podařilo splnit, ačkoliv bylo nezbytné při řešení postupné změny rychlosti rakety aktivně žákům pomáhat. Některé problémy byly zapříčiněny nedostatečnou schopností učitele řešit všechny problémy, zejména pak opomenutím nezbytné části konfigurace objektu rakety, ale i potížím při implementaci zpomalování rakety a jejího následného pádu. Aktivita byla v ostatních ohledech koncipovaná správně a vyžaduje tak spíše komplexnější přípravu učitele, než-li úpravu předchozí aktivity. Naopak se ve třídě vyskytli dva žáci, kteří se nástroji a tvorbě aplikací v nich věnovali i mimo výuku, což nasvědčuje, že pro ně byly úlohy, nebo minimálně samo prostředí atraktivní.

15.6 Závěrečná ověřovací úloha (6. aktivita)

15.6.1 Průběh

S počátkem hodiny byli žáci seznámení s úkolem, bylo jim předáno zadání a prostřednictvím kurzu v prostředí Moodle byli odkázáni na rozpracovaný projekt pro MIT App Inventor. Žáci si soubor stáhli a importovali ho do u sebe spuštěných instancí MIT App Inventoru.

Dále na základě zadání procházeli kód rozpracované aplikace, kterou inkrementálními úpravami a aktivním testováním, jež prováděli na svých zařízeních, postupně přepracovávali do stavu, který se blížil požadavkům zadání.

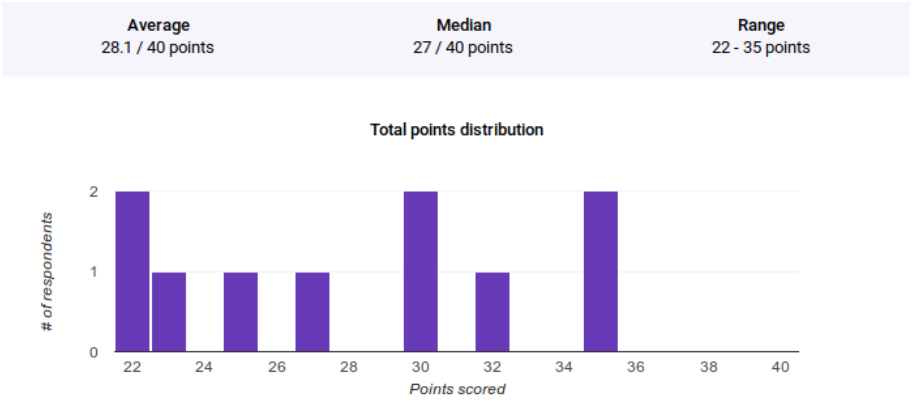
Většina žáků neměla s vymyšlením a implementací řešení větší potíže a povedlo se jim zadání vyhovět. Čtyři žáci požadavky zadání splnili s předstihem a zůstal jim i čas na vylepšení úloh nad rámec požadavků, například plynulejší pohyb kuličky či zamezení odpalu při pohybu kuličky. Naopak tři žáci nedokázali úlohu zcela vyřešit. Dva z nich realizovali pouze část řešení a tápali v předem definované struktuře zdrojového kódu, ale i v samotném prostředí MIT App Inventoru, kde měli potíže vytipovat a nalézt bloky potřebné k sestavení programu. Třetí žák měl v orientaci v předloženém programu i v prostředí MIT App Inventoru ještě větší potíže a užitečné bloky nedokázal použít ani po konzultaci s učitelem. Naopak ale správně upravil vzhled aplikace tak, aby byly všechny prvky viditelné a herní plocha zabírala celou obrazovku.

15.6.2 Postřehy žáků

Některým žákům se nedařilo importovat projekt aplikace do svého prostředí MIT App Inventoru. Důvodem by špatný název projektu, který nepovoluje speciální znaky, čísla či dokonce ani pomlčku.

15.6.3 Pohled a rozšíření druhého učitele

Pozorující učitel shledal praktické ověření za vhodné a využil ho i ke známkování své třídy. K samotnému průběhu ani k připraveným materiálům nevnesl žádnou výtku. Rovněž došlo, nad rámec této práce, k rozšíření ověření znalostí o teoretický test provedený za pomocí formulářů Google. Jeho výsledky víceméně odpovídají práci žáků v průběhu aktivit, ale třídu nijak dramaticky nerozdělují. Rovněž je důležité uvést, že zadání teoretického testu nebylo triviální a v několika úlohách vyžadovalo hlubší uvažování (např. sledování proměnné a zjištění jejího stavu po dokončení běhu kódu či porovnání dvou mírně odlišných zápisů a písemné vyjádření jejich odlišností). Test se nezabýval specifiky MIT App Inventoru, ale kladl důraz na algoritmizaci. U otázek, které se od MIT App Inventoru odvíjely, udával podstatná fakta o chování relevantních prvků pro možné řešení úlohy i bez znalostí nástroje samotného. Většinu odpovědí tvořily uzavřené otázky s výběrem, které hodnotil nástroj sám. Otevřené otázky byly hodnoceny ručně a individuálně na základě odpovědi respondenta.



Obrázek 12: Přehled výsledků původně neplánovaného teoretického ověřovacího testu

Z některých odpovědí otevřených otázek potom bylo velmi dobře patrné také to, zda žáci dávali v průběhu výuky pozor, nebo pouze bezmyšlenkovitě a se zpožděním opisovali řešení z plátna. I přes opakované užití podmínky bez explicitně uvedeného porovnání, tj. např `if nazevPromenne` na místo `if nazevPromenne == true`, pak jeden žák odpověděl, že `if nazevPromenne` je špatný zápis, neboť zde chybí porovnání. Některé odpovědi byly naopak dokonale sestaveny a zcela přesně popisovaly situaci určité úlohy.

15.6.4 Závěr

Ověřování nabytých znalostí u většiny žáků neodhalilo žádné zásadní nedostatky. U třech lidí se projevila absence zájmu, což bylo patrné i z laxnějšího přístupu k hodinám. Právě u těchto jedinců docházelo k větším komplikacím i při plnění testové úlohy, kde se žáci nedostatečně orientovali v prostředí a činilo jim problém hledání jednotlivých bloků ke skládání algoritmů. Dva z nich potom s lehkou pomocí a vytipováním některých vhodných bloků dokázali aplikace sestavit do téměř funkčního stavu. Jednomu žákovi nestačilo ale ani to.

Většina zúčastněných ale v testování obstála na výbornou a obhájili tak své znalosti tvorby mobilních aplikací, ač pouze na vytipovaném subsetu. Použity ovšem byly procedury i funkce, cykly, proměnné a podmínky. Z vlastních prvků pak zejména hodiny, Canvas a jeho vlastní prvky, rozložení stránky, tlačítko a popisek.

16 Celkové shrnutí

Kromě 3. aktivity „Hrátky s různými typy proměnných, polem a grafickým seznamem“ (viz shrnutí na straně 91) probíhalo ověřování úloh vcelku bezproblémově a v jejich průběhu nedocházelo k žádným zásadním potížím. Především z počátku byla největší překážkou špatně nastavená školní síť WiFi, v niž nedokázala mobilní aplikace MIT AI Companion vůbec pracovat. Žáci tak nemohli své výtvory ověřovat. Nepodařilo se zprovoznit ani připojení skrze USB kabel, a to i při dodržení všech bodů z návodu. Testování tak bylo z důvodu komplikací pozdrženo o celý týden, než se podařilo situaci obejít vytvořením vlastní sítě skrze přenosný WiFi router přímo pro potřeby výuky.

Zajímavostí bylo, že se zaujatější žáci snažili řešení nejrůznějších problémů promptně vyhledávat na internetu, přičemž tito dokázali z vyhledaných zdrojů sumarizovat veskrze relevantní odpovědi. To, ale i další faktory, které jsou patrné z výše uvedených popisů k jednotlivým ověřením úloh, nasvědčují poměrně velkou diversitu v rámci jedné testovací třídy. Přítomni byli žáci, kterým bylo téma blízké a měli pro něj sami přirozené nadšení, ostatním bylo potřeba v řešení úloh ve větší či menší míře pomáhat, motivovat je a určitým způsobem je do řešení některých částí úloh tlačit. Náročnost některých úloh, přesněji jejich částí, byla komplikovaná a nutila žáky více se ptát na správné řešení a dávala jim větší záminku opisovat řešení z projektoru. To však šlo také ruku v ruce se spíše instruktivistickým pojetím výuky, které jim nedávalo tolik prostoru k vlastnímu bádání.

Užitý software MIT App Inventor s sebou přinášel kromě zaznamenaných problémů především zjednodušení, které žákům většinové nástroje nenabízejí. Intuitivní tvorba prostředí a zvláště potom samotné programování díky blokovému editoru kódu jsou pro nováčky schůdným přístupem k seznámení se s algoritmizací. Také skutečnost, že MIT App Inventor, ačkoliv sám využívá blokový editor kódu ne nepodobný prostředí Scratch, nemá syntaxí daleko k čistému programování zdrojového kódu základních aplikací pro Android, posouvá sílu tohoto nástroje ještě o kus dále. Vzhledem k této vlastnosti se jedná o užitečný nástroj zvláště pro přechod mezi základy programování v blokovém editoru kódu ke strohému programování v textovém režimu.

Naopak se u tohoto softwaru žáci setkávali s různými komplikacemi, které více či méně znepříjemňovaly práci, zejména v kritickém aspektu ověřování softwaru na mobilních zařízeních. Mezi ně patřily mj. uvedené počáteční problémy s připojením k internetu (specifikum MIT App Inventoru, který pro komunikaci mezi mobilním zařízením a servery MIT využívá porty, které mohou být na některých školních ale i veřejných sítích zakázány). Dále problémy způsobené špatným rozložením stránky na displeji mobilních zařízení, konkrétně přetékáním některých prvků mimo prostor obrazovky, které se projevovalo u zhruba poloviny žáků. V těchto případech bylo chybu možné obejít změnou užitých jednotek. Projevovaná chyba však nedávala smysl a nepodařilo se vyhledat její správné řešení. V neposlední řadě potom docházelo k různým pádům mobilní aplikace MIT AI2 Companion v průběhu užíváni či nutnosti aplikaci restartovat pro projevení změn, což byly ale snadno a rychle řešitelné situace.

Z praktického ověření lze ohodnotit použitý nástroj MIT App Inventor jako velmi vhodný. Obrovským kladem je například skutečnost, že je zcela nezávislý na hostitelském počítači, díky čemuž je možné na práci pokračovat odkudkoliv. Žáci tak mohou na svých projektech pracovat nejen ve škole, ale i z domova. Nejsou limitování výukou a mohou vytvářet své vlastní aplikace, čehož část žáků zcela samostatně využila. Rovněž během výuky mohou sedět pokaždé u jiného počítače, aniž by o svá data přišli, nebo je museli libovolným způsobem mezi počítači sdílet. Podobně důležitá je

i skutečnost, že většinu žáků volba nástroje neomezuje ani v testování aplikací, neboť mobilní zařízení s operačním systémem Android je nejen na školách majoritně, s velkým odstupem rozšířený. (*Mobile Operating System Market Share Worldwide*)

Žáci si prošli vybranými možnostmi MIT App Inventoru a všemi základními algoritmickými konstrukcemi, včetně práce s podmínkami, cykly, proměnnými, nevyhnuli se ani prací s různými datovými typy a poli a rovněž nejrůznějším matematickým operacím. S jejich užitím poté, při začlenění zejména posluchačů událostí umožnujících realizaci interaktivity, dokázali vytvářet nejrůznější aplikace a hry. V průběhu ověřování úloh předváděli aktivnější žáci vyučujícímu různé své dobrovolně vytvářené domácí práce, ale i vylepšené realizace zadaných úloh, čímž potvrdili, že byly nástroj i realizované náměty dostatečně motivační. Nabyté znalosti pak byly ověřeny v závěrečné úloze, která prokázala, že si většina žáků z realizovaných aktivit odnesla mnoho požadovaných vědomostí. Žáci si až na jednu výjimku nástroj osvojili natolik, že práci na testové úloze zvládali bez větších potíží. Rovněž si zapamatovali důležité postupy, s nimiž se předem, v průběhu výuky, sami setkali.

Ačkoliv testovací vzorek nebyl příliš obsáhlý a výzkum neprobíhal na více školách, podařilo se celou sestavu úloh zdárně otestovat. Průběh práce na většině aktivit ale probíhal víceméně bezproblémově a pouze v několika málo případech by bylo vhodné přehodnotit obtížnost některých jejich částí, případně vylepšit výklad učitele a poskytnou více času na pochopení problému. S přihlédnutím ke zkušenostem z výuky, hodnocení žáků a pozorujícího učitele a především pak k výsledkům závěrečné ověřovací úlohy je možné označit testování za úspěšné. Žádné z komplikací, které se během výzkumu objevily, neměly na další průběh zásadní negativní dopad. Jednotlivé úlohy byly ohodnoceny a v problémových případech vzneseny návrhy na změnu, které se snaží zjištěné potíže adresovat.

Tabulka 22: Průměrné hodnocení všech aktivit (zaokrouhleno)

aktivita	**porozumění zadání**	**náročnost**	**atraktivita tématu**
1.	1,7	1,8	1,8
2.	1,36	2,18	1,54
3.	2,32	2,5	2,54
4.	2,94	3,1	2,1
5.	1,91	2,1	1,41
vážený průměr (zaokrouhleno na 2 desetinná místa)	2,04	2,34	1,87

Z jednotlivých dotazníků vycházejí nejhůře 3. a 4. úloha. Za podprůměrné hodnocení 3. úlohy může zejména nižší atraktivita tématu (nejnižší ze všech hodnocených úloh). Úloha byla vzhledem ke své vysoké náročnosti špatně zařazena/navržena a bylo by vhodné provést u ni revizi, viz související závěr uvedený na straně 92. Ani 4. aktivita, která však žákům alespoň na základě přímé odezvy v rámci výuky i pozorování druhého učitele nečinila žádné zásadní potíže, se nesetkala s příliš pozitivním hodnocením. Ačkoliv atraktivita tématu byla hodnocena lépe, než-li v úloze předchozí, porozumění zadání i náročnost se setkala s dalším propadem.

Na základě závěrečné ověřovací aktivity (strana 99) je potom patrné, že si žáci, až na jednu výjimku,

práci s prostředím editoru osvojili a naučili se v něm vytvářet a upravovat různé typy algoritmických konstrukcí.

16.1 Hodnocení všech aktivit zúčastněnými žáky

V závěrečném hodnocení, které se týkalo všech aktivit za celou dobu ověřování, pak dopadly úlohy poměrně dobře. Žáci je hodnotili ve srovnání s průměrem vypočítaným z jednotlivých hodnocení (na straně 102) nadmíru pozitivně. Z komentářů, které někteří žáci v celkovém hodnocení poskytli, se neobjevily žádné nové podnětné informace.

Tabulka 23: Záznam pozorovatele (přehled za všechny aktivity)

Obliba témat	**Návrh na změnu**
1	Hrál CSKO/LOLKO atd...
1	Jednodušší úlohy
1	Víceméně mi všechno vyhovovalo
1	
1	
2	Asi nic, aktivity se mi líbily, jen se párkrát stalo, že jsem tomu úplně nerozuměl
2	Mně se to líbilo a nejspíš bych nic neměnil
2	
2	
4	

16.2 Hodnocení všech aktivit druhým učitelem

Z ústního rozhovoru s druhým vyučujícím, který byl po celou dobu ověřování jednotlivých aktivit během výuky přítomen a sledoval jejich průběh, je možné zdůraznit především problémové zařazení práce s dvojrozměrným polem současně při seznamování se s konceptem proměnná a datový typ. Tento problém byl již dříve odhalen a v budoucnosti by bylo lepší úlohu koncipovat jinak, případně ji zcela vyřadit a její části zakomponovat do úlohy předcházející a následující, viz letmý návrh alternativní kompozice na straně 93. Vyučující neměl žádné další podněty k výuce a byl spokojený s tím, že se nakonec povedly realizovat všechny aktivity a některé žáky motivovaly natolik, aby s nástrojem pracovali i doma na vlastních nápadech. Rovněž se na základě osobního rozhovoru se žáky, který proběhl po dokončení všech aktivit, dozvěděl, že je práce v MIT App Inventoru bavila a náměty se jim líbily.

Část III

Závěr

Diplomová práce rozkryla téma algoritmického myšlení a způsobů jeho výuky. Nedílnou součástí byla také analýza současného stavu výuky algoritmického myšlení na středních školách. V tomto směru se podařilo shromáždit potřebná data, na jejichž základě bylo zjištěno, že se problematice výuky algoritmického myšlení určité procento škol do značné míry věnuje, ovšem velká část tomuto tématu zatím nepřikládá velkou váhu. Teoretická část je uzavřena přehledem možných programovacích přístupů, prostředků pro tvorbu i cílové nasazení programu a výběrem vhodného nástroje pro použití s mobilními dotykovými zařízeními. Na základě provedeného výzkumu byl pro potřeby praktického ověření zvolen nástroj MIT App Inventor a související cílová mobilní platforma Android.

Hlavním cílem diplomové práce bylo zjistit, zda mohou mobilní dotyková zařízení a jejich využití při výuce algoritmizace v edukačním procesu pomoci a navrhnout a následně ověřit možné úlohy pro osvojení algoritmického myšlení. Z tohoto důvodu byl pro ověření zvolen akční výzkum, jež byl realizován na Gymnáziu dr. Václava Šmejkala v Ústí nad Labem a jehož se účastnilo celkem 13 žáků. Při výzkumu byli přítomni dva vyučující, z toho jeden figuroval pouze jako pozorovatel. V průběhu dílčích aktivit byla postupně sbírána zpětná vazba od všech žáků i od pozorujícího vyučujícího. Po dokončení všech úloh, tj. v závěru na konci výzkumu, odevzdali závěrečné souhrnné hodnocení. Výuku samotnou vedl autor této práce. Celkově tak ke každé úloze i v závěrečném shrnutí bylo využito jak přímého pozorování, tak i odezvy od přítomných žáků a pozorujícího učitele.

V práci se podařilo naplnit všechny cíle. Požadavky cíle C1 byly naplněny analýzou nalezených zdrojů a rozpracovány v kapitolách Algoritmus a Algoritmické myšlení. Za Algoritmické myšlení se na základě zkoumaných pramenů považuje způsob řešení algoritmicky řešitelných problémů jejich rozložením na menší celky až jednotlivé části. Je tak úlohy možné řešit logickým rozdělením na jednotlivé kroky, které, pokud jsou korektní, dovedou při jejich přesném dodržením řešitele vždy ke správnému výsledku.

Cíli C2 bylo vyhověno analýzou současného stavu výuky algoritmického myšlení na školách v podkapitole Stav výuky algoritmického myšlení u nás a ve světě. Bylo zjištěno, že navzdory početnému zastoupení českých škol, které toto téma stále opomíjí, je výuka algoritmického myšlení i přes nedostatečné zakotvení v rámcových vzdělávacích programech na mnoha školách naplňována. Zajímavým zjištěním rovněž byl výsledek průzkumu, který odhaloval mj. rozličné způsoby přístupu jednotlivých škol k této problematice a to zejména ve zvolených metodách.

Následoval rozbor a analýza požadavků pro komplexní cíl C3 a to v kapitolách Možné prostředky pro výuku algoritmizace, Programovací jazyky, Tvorba mobilních aplikací a Výběr vhodného nástroje. Součástí byl především pohled na možné kategorie zařízení pro podporu rozvoje algoritmického myšlení, ze kterého vyplynuly poměrně široké možnosti, jež jsou školám k dispozici. V této souvislosti bylo dále rozkryto i téma programovacích jazyků, byl proveden náhled na jejich dostupnost na jednotlivých zařízeních a analyzovány dva typy způsobů programování, z nichž pro nováčky vycházejí grafické (blokové) programovací jazyky jako nejvhodnější. Neméně důležitým tématem v souvislosti se zaměřením na mobilní zařízení byl také přehled aktivně se vyvíjejících mobilních

operačních systémů, dostupných vývojových prostředí a následný výběr konkrétního vhodného nástroje (s podporou programování v blocích) pro realizaci praktické části práce.

Dokončením akčního výzkumu zahrnujícího výuku, pozorování a sběr dat s využitím dotazníků byl v závěru splněn rovněž cíl C4, k němuž byly zpracovány podklady v části Akční výzkum. Jeho hlavní náplní byl návrh a realizace aktivit s cílem rozšířit znalosti žáků v oblasti algoritmického myšlení. Ty byly následně ověřeny v praxi a na základě přímého pozorování výuky dvěma učiteli, pravidelné zpětné vazby od žáků i závěrečné ověřovací úlohy byly v kapitole Ověření úloh v praxi i s těmito údaji shrnuty reflektivní závěry a případné náměty na úpravu v možné další iteraci testování daných úloh.

Na základě realizovaného výzkumu bylo potom ověřeno, že jsou mobilní dotyková zařízení skutečně vhodnými prostředky pro výuku algoritmizace žáků. Vlastní aplikace na vlastních přístrojích jsou totiž pro žáky atraktivní zvláště tím, že si je mohou odnést domů, případně je neprodleně ukázat vyučujícímu, kamarádům, ale i rodičům. Nezanedbatelnou skutečností je rovněž snadná obsluha skrze dotykový displej a nepřeberné množství možností, které mobilní zařízení s celou řadou zabudovaných senzorů, online konektivitou ale především i celou škálou dostupných silných programovacích jazyků naskýtají. Zvolené online prostředí MIT App Inventor jim navíc umožňuje pracovat na rozpracovaných či zcela nových projektech odkudkoliv (nikoliv pouze ve škole) a věnovat se tak algoritmickým záležitostem dále i mnohem více do hloubky, čehož byli aktivnější žáci důkazem.

Zjištěné výsledky výzkumu mohou dále sloužit pro vylepšení připravených aktivit, případně posloužit jako východisko pro tvorbu podobně orientovaných jiných aktivit. Pro budoucí praxi jsou zejména užitečná zjištění problémových částí, jejichž znalost může pomoci předejít jejich opakování.

Použitá literatura a prameny

About Us [online]. Somerville, Massachusetts (USA): Arduino LLC, c2019 [cit. 2019-04-04]. Dostupné z: https://www.arduino.cc/en/Main/AboutUs.

Akční výzkum ve škole. Praha: UK PedF, 2003. Č. 3. ISSN 0031-3815. Dostupné také z: http://pages.pedf.cuni.cz/pedagogika/?p=1942&lang=cs.

Algoritmus [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2018-07-20]. Dostupné z: https://cs.wikipedia.org/wiki/Algoritmus.

Android (operating system) [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-03-27]. Dostupné z: https://en.wikipedia.org/wiki/Android_(operating_system).

Android Open Source Project [online]. Mountain View (California): Google, Open Handset Alliance [cit. 2019-04-04].

Android Studio [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2018-08-31]. Dostupné z: https://en.wikipedia.org/wiki/Android_Studio.

App development [online]. Berlín: UBports Foundation, c2019 [cit. 2019-03-27]. Dostupné z: http://docs.ubports.com/en/latest/appdev.

App Inventor Classic Reference Documentation [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/content/reference-documentation.html.

App Inventor for Android [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2018-08-03]. Dostupné z: https://en.wikipedia.org/wiki/App_Inventor_for_Android.

App Inventor for Educators [online]. Massachusetts: Massachusetts Institute of Technology, c2019 [cit. 2019-03-27]. Dostupné z: http://teach.appinventor.mit.edu.

App Inventor Tutorials [online]. Uvita (Kostarika): Pura Vida, c2018 [cit. 2019-03-27]. Dostupné z: https://puravidaapps.com/tutorials.php.

App of the Month Winners [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/app-month-gallery.html.

Arduino [online]. Somerville, Massachusetts (USA): Arduino LLC, c2019 [cit. 2019-04-04]. Dostupné z: https://www.arduino.cc.

BAJTLER, Martin. *Školy informatiku neřeší, říkají zakladatelky kurzů programování pro děti* [online]. Praha: IDNES.cz, 2017 [cit. 2018-07-27]. Dostupné z: https://praha.idnes.cz/programovani-informatika-skola-kurz-klara-stouracova-silvie-zemanova-deti-1b4-/praha-zpravy.aspx?c=A171020_359319_praha-zpravy_rsr.

BALANSKAT, Anja; ENGELHARDT, Katja; LICHT, Alexandra Hanna. *Strategies to include computational thinking in school curricula: in Norway and Sweden* [online]. Brusel (Belgie): European Schoolnet, 2018 [cit. 2019-03-12]. Dostupné z: http://www.eun.org/documents/411753/817341/Computational_thinking_report_2018.pdf/4d3d6fa0-dedd-4b62-a201-a26bf4dfd3a0.

Baltík [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/Balt%C3%ADk.

Běhové prostředí [online]. San Francisco (CA): Wikimedia Foundation, 2001-2017 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/B%C4%9Bhov%C3%A9_prost%C5%99ed%C3%AD.

Blockly [online]. Mountain View (California): Google, 2019 [cit. 2019-03-31]. Dostupné z: https://developers.google.com/blockly.

Bobřík informatiky [online]. České Budějovice: Pedagogická fakulta Jihočeské univerzity v Českých Budějovicích, c2008-2018 [cit. 2019-04-04]. Dostupné z: https://www.ibobr.cz.

Books [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/books.html.

C (programming language) [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-04-04]. Dostupné z: https://en.wikipedia.org/wiki/C_(programming_language).

C++ [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-04-04]. Dostupné z: https://en.wikipedia.org/wiki/C++.

Comparison of Mobile App Frameworks for Cross-Platform Development in 2019 [online]. Vídeň: Felgo, 2019 [cit. 2019-03-27]. Dostupné z: https://felgo.com/mobile-app-framework-comparison.

Connect your Phone or Tablet over WiFi [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/ai2/setup-device-wifi.html.

Connecting to a phone or tablet with a USB cable [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://appinventor.mit.edu/explore/ai2/setup-device-usb.html.

CPU Architecture: Arm CPU Architecture [online]. Cambridge: Arm Limited, c1995-2019 [cit. 2019-03-27]. Dostupné z: https://developer.arm.com/architectures/cpu-architecture.

CSIZMADIA, Andrew; CURZON, Paul; DORLING, Mark; HUMPHREYS, Simon; NG, Thomas; SELBY, Cynthia; WOOLLARD, John. *Computational thinking: A guide for teachers*. Swindon (Anglie): Computing at School, 2015. Dostupné také z: https://community.computingatschool.org.uk/resources/2324/single.

CURZON, Paul. *Algorithmic Thinking* [online]. Praha: Dan Lessner, 2019 [cit. 2019-03-25]. Dostupné z: https://teachinglondoncomputing.org/resources/developing-computational-thinking/algorithmic-thinking.

ČERNÝ, Michal. *Výukoví roboti: nástroj pro rozvoj algoritmického myšlení* [online]. Praha: RVP.CZ, 2015 [cit. 2018-07-20]. Dostupné z: https://clanky.rvp.cz/clanek/k/z/19905/VYUKOVI-ROBOTI-NASTROJ-PRO-ROZVOJ-ALGORITMICKEHO-MYSLENI.html.

DAVIS, Vicki. *15+ Ways of Teaching Every Student to Code (Even Without a Computer)* [online]. Marin County (California): Edutopia, 2018 [cit. 2018-10-11]. Dostupné z: https://www.edutopia.org/blog/15-ways-teaching-students-coding-vicki-davis.

Deklarativní programování [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2018-08-29]. Dostupné z: https://cs.wikipedia.org/wiki/Deklarativn%C3%AD_programov%C3%A1n%C3%AD.

ENDRŠTOVÁ, Michaela. *Nový způsob výuky informatiky nemá děti učit Word a Excel, ale programování a stavění robotů. Učitelé se ho ale bojí* [online]. Praha: Hospodářské noviny, c1996-2019 [cit. 2019-03-25]. Dostupné z: https://archiv.ihned.cz/c1-66384790-novy-zpusob-vyuky-informatiky-nema-deti-ucit-word-a-excel-ale-programovani-ci-staveni-robotu-zatim-ale-u-ucitelu-budi-spis-obavy.

ESP8266 [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-04-04]. Dostupné z: https://en.wikipedia.org/wiki/ESP8266.

File Extension [online]. Uvita (Costa Rica): Pura Vida Apps [cit. 2019-04-04]. Dostupné z: https://puravidaapps.com/file.php.

Freesound [online]. Barcelona: Universitat Pompeu Fabra, 2019 [cit. 2019-04-04]. Dostupné z: https://freesound.org/.

FUTSCHEK, Gerald. *Algorithmic Thinking: The Key for Understanding Computer Science* [online]. Vídeň: Vienna University of Technology, 2006 [cit. 2019-03-25]. Dostupné z DOI: 10.1007/11915355_15.

HROMKOVIČ, Juraj; KOHN, Tobias; KOMM, Dennis; SERAFINI, Giovanni. *Examples of Algorithmic Thinking in Programming Education* [online]. Riga (Litva): Olympiads in Informatics, 2016 [cit. 2018-07-20]. Dostupné z DOI: 10.15388/ioi.2016.08.

HRONOVÁ, Markéta; SKOUPÁ, Adéla. *Děti se budou učit programovat. Novinka ve výuce má být povinná už od první třídy* [online]. Praha: Hospodářské noviny, 2015 [cit. 2018-07-27]. Dostupné z: https://archiv.ihned.cz/c1-64734570-deti-se-budou-ucit-programovat-novinka-ve-vyuce-ma-byt-povinna-uz-od-prvni-tridy.

CHUMPIA, Erika. *Visual-based vs. Text-based programming languages* [online]. Brisbane: Coding Kids, c2017 [cit. 2019-03-27]. Dostupné z: https://www.codingkids.com.au/blog/visual-based-vs-text-based-programming-languages.

Imperativní programování [online]. San Francisco (CA): Wikimedia Foundation, 2001-2017 [cit. 2018-08-29]. Dostupné z: https://cs.wikipedia.org/wiki/Imperativn%C3%AD_programov%C3%A1n%C3%AD.

Installing and Running the Emulator in AI2 [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: https://appinventor.mit.edu/explore/ai2/setup-emulator.html.

JACEWICZ, Kris. *Geany on Ubuntu Touch device as text editor, source code editor, debugger and compiler for multiple languages* [online]. Taipei: Kris Jacewicz, 2018 [cit. 2019-03-27]. Dostupné z: http://kriscode.blogspot.com/2017/10/geany-on-ubuntu-touch-device-as-text.html.

Java [online]. Redwood City, Kalifornie (USA): Oracle, 2018 [cit. 2019-04-04]. Dostupné z: https://www.java.com.

JavaScript [online]. Mountain View, Kalifornie (USA): Mozilla Corporation, 2019 [cit. 2019-04-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

Karel (programovací jazyk) [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/Karel_(programovac%C3%AD_jazyk).

Katalog požadavků zkoušek společné části maturitní zkoušky: informatika, základní úroveň obtížnosti [online]. Praha: Centrum pro zjišťování výsledků vzdělávání, 2010 [cit. 2019-01-24]. Dostupné z: https://www.cmsps.cz/~marlib/maturita/informatika_zakladni.pdf.

Kodu Cupu Česko [online]. Pardubice, Praha: DELTA - Střední škola informatiky a ekonomie Pardubice a Vzdělávací centrum Microsoft ZŠ a MŠ Červený vrch Praha 6, 2019 [cit. 2019-04-04]. Dostupné z: http://www.koducup.cz.

Koncept STEM [online]. Praha: Národní ústav pro vzdělávání [cit. 2019-04-04]. Dostupné z: http://www.nuv.cz/p-kap/koncept-stem.

KOZUB, Martin. *Návrh JS knihovny pro tvorbu atypických forem didaktických testů* [online]. Praha, 2017 [cit. 2019-03-27]. Dostupné z: https://is.cuni.cz/webapps/zzp/detail/177177. Bakalářská práce. Karlova univerzita, Pedagogická fakulta, Katedra informačních technologií a technické výchovy.

KOZUB, Martin. *Nové Ubuntu z pohledu uživatele a vývojáře* [online]. Praha: IDNES.cz, 18. 11. 2014 [cit. 2019-03-27]. Dostupné z: https://kozub.blog.idnes.cz/blog.aspx?c=435816.

Kvalita a efektivita vzdělávání a vzdělávací soustavy ve školním roce 2017/2018. In: *Kvalita a efektivita vzdělávání a vzdělávací soustavy ve školním roce 2017/2018* [online]. 1. vyd. Praha: Česká školní inspekce, 2018, s. 237 [cit. 2019-01-27].

Kvalitativní výzkum: základní metody a aplikace. 2., aktualiz. vyd. Praha: Portál, 2005. ISBN 80-736-7040-2.

LANDA, Lev Nachmanovič. *Algoritmy a učení: kybernetika, algoritmizace a heuristika ve vyučování*. 1. vyd. Praha: Státní pedagogické nakladatelství, 1973.

LESSNER, Dan. *Střípky z konference Didinfo 2015 (2): Výuka informatiky v Polsku* [online]. Praha: Dan Lessner, 2015 [cit. 2019-03-13]. Dostupné z: http://ucime-informatiku.blogspot.com/2015/06/stripky-z-konference-didinfo-2015-2.html.

Logo (programming language) [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2019-03-25]. Dostupné z: https://en.wikipedia.org/wiki/Logo_(programming_language).

MAŇÁK, Josef; ŠVEC, Vlastimil. *Výukové metody*. Brno: Paido, 2003. ISBN 80-731-5039-5.

Matematická informatika [online]. San Francisco (CA): Wikimedia Foundation, 2001-2017 [cit. 2018-10-12]. Dostupné z: https://cs.wikipedia.org/wiki/Matematick%C3%A1_informatika.

Merkurtoys [online]. Police nad Metují: Merkurtoys s.r.o., c2016-2019 [cit. 2019-04-04]. Dostupné z: http://www.merkurtoys.cz.

Mindstorms LEGO.com [online]. Billund (Dánsko): LEGO Group, c2019 [cit. 2019-04-04]. Dostupné z: https://www.lego.com/cs-cz/mindstorms.

Minecraft Cup [online]. Praha: Microsoft, c2019 [cit. 2019-04-04]. Dostupné z: http://minecraftcup.cz.

MIT AI2 Companion [online]. Mountain View (California): Google, c2019 [cit. 2019-04-04]. Dostupné z: https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3.

MIT App Inventor: IDE [online]. Massachusetts: Massachusetts Institute of Technology, 2019 [cit. 2019-04-04]. Dostupné z: http://ai2.appinventor.mit.edu/.

MIT App Inventor Public Open Source [online]. San Francisco (CA): GitHub Inc., 2018 [cit. 2018-08-31]. Dostupné z: https://github.com/mit-cml/appinventor-sources.

MIT spin-out Thunkable hopes its drag-and-drop app builder can be a money-spinner too [online]. New York City: Oath Inc., 2016 [cit. 2018-09-03]. Dostupné z: https://techcrunch.com/2016/03/05/mit-spin-out-thunkable-hopes-its-drag-and-drop-app-builder-can-be-a-money-spinner-too.

Mobile app development [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-03-27]. Dostupné z: https://en.wikipedia.org/wiki/Mobile_app_development.

Mobile Operating System Market Share Worldwide: Feb 2018 - Feb 2019 [online]. Dublin: StatCounter, c1999-2017 [cit. 2019-03-05]. Dostupné z: http://gs.statcounter.com/os-market-share/mobile/worldwide.

NAKHLEH, Luay. *What is Algorithmic Thinking?: Algorithmic Thinking (Part 1)* [online]. Houston (USA): RICE, 2016 [cit. 2018-09-18]. Dostupné z: https://www.coursera.org/lecture/algorithmic-thinking-1/what-is-algorithmic-thinking-X7Wpl.

National curriculum in England: computing programmes of study [online]. London: GOV.UK, 2013 [cit. 2019-03-13]. Dostupné z: https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study.

Ne jen uživatelé, ale programátoři. Výuku informatiky mají změnit nové učebnice [online]. Praha: Česká televize, 2018 [cit. 2019-01-27]. Dostupné z: https://ct24.ceskatelevize.cz/domaci/2617181-ne-jen-uzivatele-ale-programatori-vyuku-informatiky-maji-zmenit-nove-ucebnice.

Nejčastější dotazy k informatice [online]. Praha: CERMAT, 2010 [cit. 2018-09-22]. Dostupné z: http://www.novamaturita.cz/nejcastejsi-dotazy-k-informatice-1404035832.html.

NĚMEC, Petr. *Naprogramujte si aplikaci pro mobilní telefon* [online]. Praha: RVP.CZ, 2014 [cit. 2018-08-03]. Dostupné z: https://spomocnik.rvp.cz/clanek/18203/NAPROGRAMUJTE-SI-APLIKACI-PRO-MOBILNI-TELEFON.html.

NEUMAJER, Ondřej. *Ivan Ryant: Nahraďte algoritmizaci systémovým přístupem!: Názor autora na výuky algoritmizace rozhodně není zcela ojedinělý.* [online]. Praha: Albatros Media a.s., 2009 [cit. 2018-09-22]. Dostupné z: http://www.ceskaskola.cz/2009/07/ivan-ryant-nahradte-algoritmizaci.html%5C#comment-3632838790966223082.

Node.js [online]. San Francisco (CA): OpenJS Foundation a The Linux Foundation, [2018] [cit. 2019-04-04]. Dostupné z: https://nodejs.org.

NodeMcu [online]. NodeMcu Team, c2014-2018 [cit. 2019-04-04]. Dostupné z: https://www.nodemcu.com.

NodeMCU Documentation [online]. USA: Read the Docs, 2018 [cit. 2018-10-11]. Dostupné z: https://nodemcu.readthedocs.io/en/master.

NOONAN, Linda. *The Growing Importance of Computer Science* [online]. Massachusetts Business Alliance for Education, 2016 [cit. 2018-10-09]. Dostupné z: https://www.mbae.org/the-growing-importance-of-computer-science.

Operační program Výzkum, vývoj a vzdělávání [online]. Praha: Ministerstvo školství, mládeže a tělovýchovy, c2017 [cit. 2019-04-04]. Dostupné z: https://opvvv.msmt.cz.

Oracle JVM Just-in-Time Compiler (JIT) [online]. Redwood City (Kalifornie, USA): Oracle, c2017 [cit. 2019-03-31]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/Oracle-JVM-JIT.html%5C#GUID-9466BE4E-E7EE-486F-9DF8-D331B316359D.

ORAM, Mike. *Is PHP relevant?* [online]. New York City: Dev.to, 2018 [cit. 2018-10-11]. Dostupné z: https://dev.to/mporam/is-php-relevant--1np.

Osobní robot [online]. Zeleneč: Profess Consulting s.r.o., c2019 [cit. 2019-04-04]. Dostupné z: http://osobnirobot.cz.

PAULENKOVÁ, Kristína. Výuka informatiky na školách se mění, zaměří se na programování. *Hospodářské Noviny* [online]. 2017 [cit. 2018-10-09]. Dostupné z: https://archiv.ihned.cz/c1-65984410-vyuka-informatiky-na-skolach-se-meni-zameri-se-na-programovani.

PAVELKOVÁ, Adéla. *Akční výzkum v pedagogickém prostředí*. Brno, 2012. Dostupné také z: https://is.muni.cz/th/yxbkp/Akcni_vyzkum_v_pedagogickem_prostredi_.pdf. Diplomová práce. Masarykova univerzita.

PECINOVSKÝ, Rudolf. *Metodika výuky programování na rozcestí* [online]. Praha: Ing. Rudolf Pecinovský, CSc., 2007 [cit. 2018-09-22]. Dostupné z: http://vyuka.pecinovsky.cz/prispevky/2007_Po_Metodika_vyuky_na_rozcesti.pdf.

PHP: Hypertext Preprocessor [online]. PHP Development Team, Zend Technologies, c2001-2019 [cit. 2019-04-04]. Dostupné z: https://www.php.net/.

POLÁK, Jan. *Pedagogické zásady a principy* [online]. Praha: ČVUT, 2012 [cit. 2018-11-23]. Dostupné z: http://skladiste.janpolak.cz/ucitelstvi/pedagogika/pedagogika-11.pdf.

Profil absolventa - IT [online]. Praha: Smíchovská Střední Průmyslová Škola, 2017 [cit. 2018-09-20]. Dostupné z: http://www.ssps.cz/candidate/read/119.

Programming Arduino Uno (ATmega386P) in assembly [online]. San Francisco (CA): GitHub, 2015 [cit. 2019-04-04]. Dostupné z: https://gist.github.com/mhitza/8a4608f4dfdec20d3879.

Programovací jazyk [online]. San Francisco (CA): Wikimedia Foundation, 2001-2017 [cit. 2018-08-29]. Dostupné z: https://cs.wikipedia.org/wiki/Programovac%C3%AD_jazyk.

Python [online]. Wilmington, Delaware (USA): Python Software Foundation, c2001-2019 [cit. 2019-04-04]. Dostupné z: https://www.python.org.

Qt Creator [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2018-08-31]. Dostupné z: https://en.wikipedia.org/wiki/Qt_Creator.

RAMBOUSEK, Vladimír. *Edukační technologie*. Praha: UK PedF, 2017.

Rámcové vzdělávací programy [online]. Praha: Národní ústav pro vzdělávání [cit. 2019-04-04]. Dostupné z: http://www.nuv.cz/t/rvp.

Rámcový vzdělávací program pro obor vzdělání 18-20-M/01 Informační technologie [online]. Praha: Národní ústav odborného vzdělávání, 2018 [cit. 2018-09-22]. Dostupné z: http://zpd.nuov.cz/RVP/ML/RVP%201820M01%20Informacni%20technologie.pdf.

Rámcovým vzdělávacím programem pro gymnázia [online]. Praha: Národní ústav pro vzdělávání, 2013 [cit. 2018-09-22]. Dostupné z: http://www.nuv.cz/file/159_1_1.

Raspberry Pi [online]. Cambridge: Raspberry Pi Foundation, [2019] [cit. 2019-04-04]. Dostupné z: https://www.raspberrypi.org.

Sailfish OS [online]. Tampere (Finsko): Jolla, c2018 [cit. 2019-04-04]. Dostupné z: https://sailfishos.org/.

Scratch [online]. Massachusetts: MIT Media Lab, 2018 [cit. 2018-12-16]. Dostupné z: https://scratch.mit.edu.

Shell (informatika). San Francisco (CA): Wikimedia Foundation, 2001-2017. Dostupné také z: https://cs.wikipedia.org/wiki/Shell_(informatika).

SCHÖN, Otakar. *Programování se má stát běžným předmětem ve škole, s výukou pomáhají roboti. Předmět naráží překvapivě na odpor rodičů* [online]. Praha: Hospodářské noviny, c1996-2019 [cit. 2019-03-25]. Dostupné z: https://archiv.ihned.cz/c1-66169570-programovani-se-ma-stat-beznym-predmetem-ve-skole-s-vyukou-pomahaji-roboti-predmet-narazi-prekvapive-na-odporu-rodicu.

SIGCSE Technical Symposium on Computer Science Education [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-03-12]. Dostupné z: https://en.wikipedia.org/wiki/SIGCSE_Technical_Symposium_on_Computer_Science_Education.

SKALKOVÁ, Jarmila. *Obecná didaktika: vyučovací proces, učivo a jeho výběr, metody, organizační formy vyučování*. 2. vyd. Praha: Grada, 2007. ISBN 978-80-247-1821-7.

Stovky českých škol zařadily do své výuky programování [online]. Praha: Ministerstvo školství, mládeže a tělovýchovy, 2016 [cit. 2018-07-27]. Dostupné z: http://www.msmt.cz/ministerstvo/novinar/stovky-ceskych-skol-zaradily-do-vyuky-programovani.

Supported browsers and versions for Blockly web? [online]. Mountain View (California): Google, 2017 [cit. 2019-04-04]. Dostupné z: https://groups.google.com/d/msg/blockly/o3pERaRQhSg/eVyheZtvAQAJ.

Supported media formats [online]. Mountain View (California): Google, 2017 [cit. 2019-04-04]. Dostupné z: https://developer.android.com/guide/topics/media/media-formats.

Supported Platforms [online]. Espoo (Finsko): The Qt Company, 2018 [cit. 2018-08-31]. Dostupné z: http://doc.qt.io/qtcreator/creator-os-supported-platforms.html.

System Requirements [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-04-04]. Dostupné z: http://explore.appinventor.mit.edu/content/system-requirements.

Školní vzdělávací program [online]. San Francisco (CA): Wikimedia Foundation, 2001-2018 [cit. 2019-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/%C5%A0koln%C3%AD_vzd%C4%9Bl%C3%A1vac%C3%AD_program.

Školská informatika končí, děti se budou učit programovat. Učitelé však nejsou na převratné změny připraveni [online]. Praha: Albatros Media a.s., c2000-2015 [cit. 2018-10-18]. Dostupné z: http://www.ceskaskola.cz/2017/06/skolska-informatika-konci-deti-se-budou.html.

The Programming Language Lua [online]. Rio de Janeiro (Brazílie): LabLua, 2018 [cit. 2019-04-04]. Dostupné z: https://www.lua.org.

THIEBAUT, Dominique. *Tutorial: Assembly Language with the Raspberry Pi* [online]. Northampton (Anglie): Clark Science Center Smith College, 2015 [cit. 2019-04-04]. Dostupné z: http://www.science.smith.edu/dftwiki/index.php/Tutorial:_Assembly_Language_with_the_Raspberry_Pi.

THOMAS, Jakita O. *ACM Digital Library* [online]. Baltimore, Maryland (USA): Association for Computing Machinery, 2018 [cit. 2019-03-12]. ISBN 978-1-4503-5103-4. Dostupné z DOI: 10.1145/3159450.3159473.

Thunkable Cross Platform [online]. San Francisco (CA): Thunkable, 2018 [cit. 2018-08-31]. Dostupné z: https://docs.thunkable.com/thunkable-cross-platform.

Tisková zpráva ke zveřejnění ilustračního testu ze zkušebního předmětu informatika [online]. Praha: CERMAT (Centrum pro zjišťování výsledků vzdělávání), c2010 [cit. 2019-04-04]. Dostupné z: https://www.novamaturita.cz/tiskova-zprava-ke-zverejneni-ilustracniho-testu-ze-zkusebniho-predmetu-informatika-1404035852.html.

Tizen [online]. San Francisco (CA): Linux Foundation, c2012 [cit. 2019-04-04]. Dostupné z: https://www.tizen.org/.

TURING, A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* [online]. 1937, roč. s2-42, č. 1, s. 230–265 [cit. 2019-03-31]. ISSN 00246115. Dostupné z DOI: 10.1112/plms/s2-42.1.230.

Tutorials for App Inventor [online]. Massachusetts: Massachusetts Institute of Technology, c2012-2019 [cit. 2019-03-27]. Dostupné z: http://appinventor.mit.edu/explore/ai2/tutorials.html.

UBports [online]. Berlin: UBports Foundation, [2017-2019] [cit. 2019-03-27]. Dostupné z: https://ubports.com.

UBports Foundation. Berlín: UBports Foundation, c2019. Dostupné také z: https://ubports.com/foundation/ubports-foundation.

Virtuální stroj [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-03-31]. Dostupné z: https://cs.wikipedia.org/wiki/Virtu%C3%A1ln%C3%AD_stroj.

Visual programming language [online]. San Francisco (CA): Wikimedia Foundation, 2001-2019 [cit. 2019-03-31]. Dostupné z: https://en.wikipedia.org/wiki/Visual_programming_language.

VLČKOVÁ, Kateřina. *Pedagogické zásady* [online]. Brno: Kpd PdF MU, 2005 [cit. 2018-11-23]. Dostupné z: https://is.muni.cz/elportal/estud/lf/ps05/mpmp071/ped_zasady.pdf.

VUORIKARI, Riina; PUNIE, Yves; CARRETERO, Stephanie; BRANDE, Lieve Van den. *DigComp 2.0: The Digital Competence Framework for Citizens* [online]. Brusel (Belgie): Joint Research Centre, 2016 [cit. 2019-03-12]. ISBN 978-92-79-58876-1. Dostupné z DOI: 10.2791/11517.

What are Computer Programming Languages? [online]. Houston (Texas): ComputerScience.org, 2018 [cit. 2018-10-11]. Dostupné z: https://www.computerscience.org/resources/computer-programming-languages.

WILLIAMS, Lawrence. *Introducing computing: a guide for teachers*. New York, NY: Routledge, [2015]. ISBN 978-113-8022-843.

Z výzev OP VVV na šablony je možné pořídit i didaktické pomůcky a IT techniku [online]. Praha: Ministerstvo školství, mládeže a tělovýchovy, c2013-2019 [cit. 2019-03-31]. Dostupné z: http://www.msmt.cz/vzdelavani/skolstvi-v-cr/nakup-pomucek-a-vybaveni-z-vy-zev-op-vvv-na-sablony.

Seznam tabulek

Seznam obrázků

Část IV

Přílohy

Tabulka 24: Výuka algoritmického myšlení na středních školách dle odpovědí učitelů jednotlivých škol

Kraj	**Město**	**Název školy**	**Ročník**[55]	**Nástroje či metody**	**Povinný předmět**
Středočeský kraj	Mělník	Gymnázium Jana Palacha	prima	Ozoboti, Scratch	ano
			sekunda	Ozoboti, Scratch	volitelný předmět programování
			1. ročník	úvod do programování s využitím jazyka C#	ano
kraj Vysočina	Jihlava	Gymnázium Jihlava	3. ročník	zápis algoritmů ve vyšším programovacím jazyce	volitelný předmět programování
Liberecký kraj	Liberec	Střední škola a Mateřská škola	2. nebo 4. ročník, podle rozvržení ročníků a časové dotace, pouze pár hodin	iniciativa učitele nad rámec RVP a ŠVP	ano
			obor Mechanik elektrotechnik 2. ročník pro nástavbové studium, jinak 2. a 3. ročník, podobor Řídící systémy konkrétněji a více do hloubky	výuka podle učebnice Algoritmizace od Pšenčíkové, dále programovatelný Merkur, LEGO, světelné tabule	ano
Pardubický kraj	Pardubice	Gymnázium Pardubice	1. a 3. ročník	vývojové diagramy, Scratch, JavaScript	ano
	Vysoké Mýto	Gymnázium Vysoké Mýto	2. ročník	základy (pravděpodobně pouze v rámci požadavků RVP)	ano
			3. ročník	programování v Pascalu	volitelný předmět programování
			4. ročník	tvorba webových stránek	volitelný předmět programování
	Česká Třebová	Gymnázium Česká Třebová	1. ročník	kreslení vývojových diagramů	ano

[55] Mezi 1. až 4. ročníky spadá rovněž kvinta až oktáva.

			-	strukturované a objektové programování v jazyce C#, inspirace ve sbírce úloh od Pavla Töpfra a Dany Töpferové	volitelný informatický seminář
Jiho-moravský kraj	Brno	Gymnázium Brno	2. ročník	Ozoboti, Micro:bit (blokově, následně Python), úvod do Pythonu	ano
			3. a 4. ročník	Programování v Pythonu	volitelný seminář programování
		AKADEMIA Gymnázium	3. ročník	hledání opakujících se vzorů, zápis algoritmů, code.org, HTML a v plánu jsou Ozoboti	ano
			2. ročník	Vymýšlení vlastních algoritmů, seznámení s existujícími se algoritmy (např. třídící algoritmy). Dále Python, code.org a HTML a CSS	ano
			3. a 4. ročník	zaměření převážně na programování	volitelný seminář
Moravsko-slezský kraj	Frýdek-Místek	Gymnázium a SOŠ	3. ročník	algoritmizace - strukturogramy (Myšlením k algoritmům, Jaromír Kukal, 1992), bez využití PC, dále programování	volitelný
Králové-hradecký kraj	Broumov	Gymnázium Broumov	1. ročník	příkazový řádek a dávkové soubory, VBA v prostředí MS Word a MS Excel	ano
			2. ročník	základy PHP a MySQL, základy programování MCU (Arduino - ESP8266 a IoT)	ano
			3. ročník	algoritmizace jako samostatné téma, výklad nezávislý na programovacím jazyce, pokračování v PHP	ano

			4. ročník	algoritmizace a programování	volitelný seminář
	Nová Paka	Gymnázium a SOŠPg	3. ročník	Hour of Code (2 až 4 vyučovací hodiny), vývojové diagramy, případně jazyk Karel	ano
				Napříč ročníky podpora na účasti v soutěži Bobřík Informatiky	
Olomoucký kraj	Přerov	Gymnázium Jana Blahoslava a Střední pedagogická škola	3. ročník	webové stránky a tvorba aplikace ve vyšším programovacím jazyce	ano
	Šumperk	Gymnázium Šumperk	1. ročník	základy, jednoduché algoritmy - programovací jazyk Karel	ano
			dříve též 3. ročník	více do hloubky	volitelný seminář programování
Karlovar-ský kraj	Mariánské lázně	Gymnázium a obchodní akademie Mariánské Lázně	1. ročník	Blockly games, tvorba aplikace či hry dle vlastního výběru ve Scratchi	ano
				C# (Arduino + součástky)	volitelný seminář programování

Seznámení se s prostředím

MIT App Inventor je nástroj pro tvorbu mobilních aplikací pro operační systém Android. Závislost na platformě je dána produkováním nativního kódu, tedy JAVA aplikací, využívajících standardních Android knihoven.

Práce s MIT App Inventoru nicméně nevyžaduje žádné znalosti programovacího jazyka JAVA, neboť tvorba aplikací probíhá skrze dvě víceméně „klikací" prostředí.

Jedná se v první řadě o režim návrháře, kde je cílem navrhnout vzhledovou stránku aplikace z pohledu rozvržení, ale také osadit aplikaci použitými i neviditelnými komponenty, s nimiž budeme dále pracovat.

Dále je k dispozici programovací prostředí ne nepodobné Scratchi, jež sestává z palety nástrojů kategorizované podle typu činnosti, přičemž pod každou kategorií je k dispozici seznam bloků, jež je možné skládat jako puzzle. Rovněž je vypsán seznam komponent, použitých v návrháři, jež každé obsahují své metody pro získávání či nastavování vlastních hodnot.

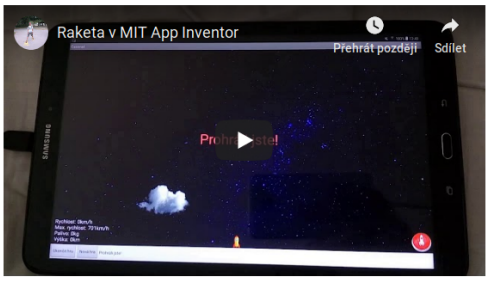
MIT App Inventor (IDE - Integrated Development Environment)

K tomu, abyste mohli s nástrojem pracovat, je nezbytné přihlásit se účtem Google.

Dokumentace

Zdrojové kódy projektu (Apache License v2)

V závěru si vytvoříte hru s raketou. Může vypadat například takto ...



V náplni jsou ale i herní, praktičtější aktivity.

Začínáme

První aplikace

Hra v canvasu a senzor orientace pohybu

Pohyb kuličky po obrazovce

Pohyb kuličky po obrazovce - Popis řešení (v .pdf)

Proměnná, pole, grafický seznam

Příprava hlavní obrazovky pro další aktivity

Příprava hlavní obrazovky - Popis řešení (v .pdf)

Záznam zvuku, ukládání, práce se souborem a databází

Záznam zvuku s možností přehrávání

Záznam zvuku - Popis řešení (v .pdf)

Časovač, canvas

Hra s raketou

Hra s raketou - Popis řešení (v .pdf)

Závěrečná ověřovací aktivita

Opravte hru

Rozbitá hra Golf

Minigolf - zadání v pdf

Test z MIT App Inventoru

Obrázek 13: Ukázka připraveného kurzu Moodle

Záznam průběhu aktivity pro vyučujícího

Přímé pozorování průběhu aktivity

Výskyty problémů žáků:

Technické aspekty |

Algoritmické konstrukce |

Pochopení problému |

Pozitivní aktivita žáků:

Technické aspekty |

Algoritmické konstrukce |

Pochopení problému |

Na co se žáci ptají? (komentáře a postřehy žáků)

Jak moc je potřeba žákům pomáhat? Co zvládnou sami? Tj. co je náročné a co jednoduché …

Je postup zvolený pro splnění aktivity vhodný?

Je vyhrazená časová dotace adekvátní pro splnění aktivity?

Průběžný anonymní dotazník pro žáky

Ohodnoť každou otázku na stupnici 1 až 5, zakroužkuj zvolenou variantu

Vyjádři číselně, jak jsi rozuměl(a) zadání?

1 – rozuměl(a) jsem všemu, 5 – nebyl(a) jsem v obraze

1 2 3 4 5

Vyjádří číselně, jak byl úkol náročný?

1 – úloha byla jednoduchá, 5 – vůbec jsem se v tom neorientoval(a)

1 2 3 4 5

Vyjádři číselně, jak moc se ti téma líbilo?

1 – úloha mě nadchla, 5 – téma bylo strašné, vymyslel(a) bych mnohem zajímavější

1 2 3 4 5

Napiš mi vlastní názor k tomu, co jsme dělali:

Závěrečný anonymní dotazník pro žáky

Vyjádři číselně, jak moc se ti témata celkově líbila?

1 – úlohy mě nadchly, 5 – témata byla otřesná, vymyslel(a) bych mnohem zajímavější

1 2 3 4 5

Co bys dělal(a) jinak, co bys změnil(a)?

Univerzita Karlova, Pedagogická fakulta
M. Rettigové 4, 116 39 Praha 1

Evidenční list žadatelů o nahlédnutí do listinné podoby práce

Jsem si vědom/a, že závěrečná práce je autorským dílem a že informace získané nahlédnutím do zveřejněné závěrečné práce nemohou být použity k výdělečným účelům, ani nemohou být vydávány za studijní, vědeckou nebo jinou tvůrčí činnost jiné osoby než autora.

Byl/a jsem seznámen/a se skutečností, že si mohu pořizovat výpisy, opisy nebo rozmnoženiny závěrečné práce, jsem však povinen/povinna s nimi nakládat jako s autorským dílem a zachovávat pravidla uvedená v předchozím odstavci tohoto prohlášení.

Poř. č.	Datum	Jméno a příjmení	Adresa trvalého bydliště	Podpis
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				
10.				